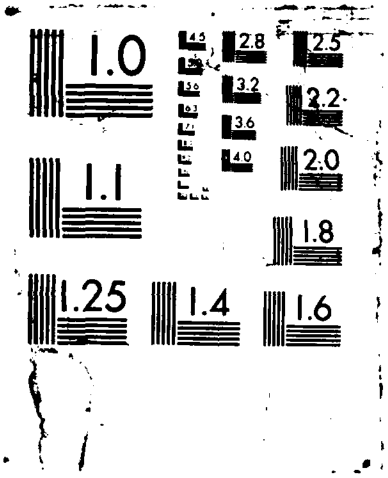
AD-A190 010 METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION VOLUME 1/3
1(U) SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO
CA J MCCALL ET AL. NOV 87 RADC-TR-87-171-VOL-1
UNCLASSIFIED F30602-83-C-0118 F/G 12/5 NL

6


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

4

RADC-TR-87-171, Vol I (of two)
Final Technical Report
November 1987

METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION

Science Applications International Corporation

J. McCall, W. Randall, C. Bowen, N. McKelvey, R. Senn, J. Morris, H. Hecht, S. Fenwick, P. Yates, M. Hecht and R. Vienneau

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS
N/A

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
N/A

5. MONITORING ORGANIZATION REPORT NUMBER(S)
RADC-TR-87-171, Vol I (of two)

6a. NAME OF PERFORMING ORGANIZATION
Science Applications International Corporation

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Rome Air Development Center (COEE)

6c. ADDRESS *(City, State, and ZIP Code)*
10260 Campus Point Drive
San Diego CA 92121

7b. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Rome Air Development Center

8b. OFFICE SYMBOL *(If applicable)*
COEE

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F30602-83-C-0118

8c. ADDRESS *(City, State, and ZIP Code)*
Griffiss AFB NY 13441-5700

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.
62702F	5581	20	43

11. TITLE *(Include Security Classification)*
METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION

12. PERSONAL AUTHOR(S) J. McCall, W. Randall, C. Bowen, N. McKelvey, R. Senn, J. Morris, H. Hecht, S. Fenwick, P. Yates, M. Hecht, R. Vienneau

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM Jun 83 TO May 87

14. DATE OF REPORT *(Year, Month, Day)*
November 1987

15. PAGE COUNT
208

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	
12	08	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Software Reliability
Software Reliability Engineering
Software Measurement

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report describes the results of a research and development effort to develop a methodology for predicting and estimating software reliability. A Software Reliability Measurement Framework was established which spans the life cycle of a software system and includes the specification, prediction, estimation, and assessment of software reliability. Data from 59 systems, representing over 5 million lines of code, were analyzed and generally applicable observations about software reliability were made. A detailed approach to the collection and analysis of reliability data is also presented.

Participating in the effort were the following: 1) Science Applications International Corporation (J. McCall, W. Randall, S. Fenwick, C. Bowen, P. Yates, N. McKelvey); 2) SoHar, Inc. (M. Hecht, H. Hecht); and, 3) ITT Research Institute (R. Senn, J. Morris, R. Vienneau).

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED □ SAME AS RPT □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Joseph P. Cavano

22b. TELEPHONE *(Include Area Code)*
(315) 330-4063

22c. OFFICE SYMBOL
RADC (COEE)

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

PREFACE

The variation in fault density on Air Force programs is enormous: the worst programs are 390 times more error-prone than the best. Obviously, there are some critical differences in these programs that cause more errors to be introduced or left undetected. If we could solve the problem of what these differences are and how to control them, then we would have learned something fundamental about the occurrence of errors in software and how to avoid them.

To increase our understanding of what happens during a software project, this effort sought to discover empirical evidence of development process and software product variables that affect error occurrence. The starting point was a set of variables characterizing software quality that were developed in previous RADC work. RADC used three methods to gather data: reviewing published reports, examining software error data bases from the NASA Software Engineering Laboratory and the RADC Data and Analysis Center for Software, and collecting information directly from three software projects. RADC analyzed 59 projects, totaling over 5 million lines of code, to refine the initial set of variables and obtained sufficient evidence to recommend 8 variables for use in controlling software errors.

Using these variables, RADC developed prediction and estimation models to express software reliability in terms of fault density (the number of faults per executable lines of code) and failure rate (the number of failures during the execution time of a program). Through the prediction and estimation techniques, project personnel can see what variables affect fault density and failure rate and can determine what variables can be controlled in their projects to meet requirements. During an experimental application of the predictive and estimation techniques, there was less than a 20% error between the values predicted by the techniques and what actually occurred on a small Production Center-type application. Although the techniques are by no means validated, this result is encouraging.

In addition to the predictive techniques, RADC developed checklists that could be applied throughout the life cycle to help improve the quality of the software. The checklists are a series of questions to be answered at key milestone reviews. Detailed procedures were also produced to show how to measure the variables and apply the checklists and are available in the guidebook companion to this volume.

Joseph P. Cavano
Project Engineer

VOLUME I
TABLE OF CONTENTS

SECTION PAGE

VOLUME I
TABLE OF CONTENTS (CONTINUED)

SECTION PAGE

VOLUME I
TABLE OF CONTENTS (CONTINUED)

VOLUME I
LIST OF TABLES

VOLUME I
LIST OF TABLES (CONTINUED)

VOLUME I
LIST OF FIGURES

VOLUME I
LIST OF FIGURES (CONTINUED)

VOLUME I
LIST OF FIGURES (CONTINUED)

1.0 INTRODUCTION

1.1 PURPOSE

The purpose of this report is to describe the results of a research and development effort to develop a methodology for predicting and estimating software reliability. This report represents the final report of the project. This effort was performed under Contract Number F30602-83-C-0118 for the U.S. Air Force Rome Air Development Center (RADC).

1.2 SCOPE

The reliability of computer-based systems (particularly embedded systems) within the Department of Defense (DoD) has been a subject of considerable concern for a number of years. For most DoD systems, the reliability of the system is critical to effective mission performance. In the past, the approach to determining or predicting system reliability has been to look at the hardware components, calculate their combined reliability, assume software reliability was one, and use the hardware reliability number as the system reliability.

Experience, however, has shown that software is a significant contributor to system failures. In fact, the reliability of hardware components in Air Force computer systems has improved to a point where software reliability is becoming the major factor in determining the overall system reliability. Hardware reliability is a well-understood aspect of system engineering, with measures for Mean-Time-Between-Failures and a model dealing with the aging of components.

Software reliability is a more complex concept than hardware reliability and is not understood nearly as well. Attempts to predict software reliability have met with limited success. Without an accepted predictive software reliability figure-of-merit and/or software reliability estimation number, it is impossible to determine the impact of software reliability on system reliability. This effort seeks to improve reliability prediction and estimation.

Since 1976, RADC has been pursuing a program to achieve better control of software quality. The thrust has been threefold. One dimension of the research centers around an RADC and Electronic Systems Division sponsored effort entitled, "Factors in Software Quality" [MCCA77], which established a three-level hierarchical framework of software quality and determined that software quality can be measured and predicted by the absence, presence, or degree of some identifiable software product attributes. At the top level of the framework, user-oriented factors that contribute to software quality have been defined (including reliability, correctness, testability, maintainability, flexibility, integrity, reusability, etc.). These factors were succeeded by more software-oriented criteria and metrics at the second and third levels, respectively. Additional research sponsored by RADC and the U.S. Army Computer Systems Command has: (1) enhanced this framework, and (2) developed an Automated Quality Measurement System (AMS). This work is related to those efforts by seeking to improve and enhance the measurement of software reliability. The results of the above efforts have been documented in:

- "Software Reliability Study", RADC-TR-76-238 [THAY76],
- "Factors in Software Quality", RADC-TR-77-369 [MCCA77],
- "Software Quality Metrics Enhancement", RADC-TR-80-109 [MCCA80]
- "Software Quality Measurement for Distributed Systems", RADC-TR-175 [BOWE83], and
- "Specification of Software Quality Attibutes", 3 Volumes, RADC-TR-85-37 [BOWE85].

The RADC Quality Measurement Framework identifies four factors that impact software and system reliability:

1. Software Reliability (the extent to which a program can be expected to perform its intended function with required precision).
2. Software Correctness (the extent to which a program satisfies its specifications and fulfills the user's mission objectives).
3. Software Maintainability (the effort required to locate and fix an error in an operational program).
4. Software Testability (the effort required to validate the specified software operation and performance).

These factors and their associated criteria and metrics attempt to predict software performance by measuring various attributes from software code and documentation such as the software's consistency, completeness, simplicity, accuracy, error tolerance, modularity, etc. The measurements can be taken across the software development life-cycle so that an early determination of these qualities can be made.

A second dimension of the research is reliability models. RADC has been active in developing and validating software reliability estimation models such as the Imperfect Debugging Model, the Non-homogeneous Poisson Process Model, the IBM Poisson Model and the Generalized Poisson Model [GOEL83]. These models analyze failure data from software testing in order to estimate the total number of software errors present and the rate of occurrence at which the errors are being exposed. The models generally define a Mean-Time-Between-Failures (MTBF) based on the failure data analysis.

An RADC-sponsored survey lists 24 quantitative software reliability models that have been published up to 1979 [DACS79]. Of those, 19 were primarily useful for estimation and five (5) were primarily useful for prediction. All except one (1) of the latter predicted an initial (usually interpreted to mean at start of formal test) error content, and by the relations discussed below, this could be translated into a failure rate and thus be transitioned into an estimation model.

Practically all of these models assume:

- A fixed initial number of faults (bugs);
- A failure rate of probability that is positively correlated with the number of faults; and
- The number of faults will be reduced as failures are observed (not necessarily on a one-to-one basis).

In the simplest case, the failure rate is proportional to the number of faults, decreases by one for every failure that is observed, and no new faults are introduced during the correction. The failure rate is designated by u(t) and the number of faults by E(t). Then

$$u(t) = k\ E(t), \qquad (1)$$

where k is the constant of the proportionality. At start of formal test,

$$u(0) = k\ E(0), \qquad (2)$$

and after an arbitrary number of failures, C, have been observed (by our assumptions exactly C faults have, therefore, been removed) and the failure rate is

$$u(1) = k\ [E(0) - C]. \qquad (3)$$

Since u(0), u(1), and C are known, k and E(0) can be computed as

$$k = [u(0) - u(1)]/C \qquad (4)$$

and $$E(0) = u(0)\ C/[u(0) - u(1)] \qquad (5)$$

Thus, the initial fault content and the number of remaining faults can be obtained. Also, because the failure rate corresponds to the fault removal rate

$$u = -dE/dt, \qquad (6)$$

which can be combined with eq.(1) to yield

$$E(t) = E(0) \exp(-kt) \qquad (7)$$

In other words, the fault content of a program and the failure rate both approach zero exponentially. The relations outlined here can be used primarily for reliability estimation. It is generally agreed that at the start of formal test about one percent of all statements contain a fault [MORA76]. This was also observed in [FISH79]. If the length of a program (and hence the initial fault content) is known, this can be used to predict the initial failure rate through use of eq.(2), and the failure rate at any other time by adding the relation in eq.(7). Estimation can be based simply on eq.(7) which permits translating the failure rate at one time into the failure rate at another (future) time.

Many of the models described in [DACS79] allow for imperfect debugging (not every failure results in a fault removal, and some corrections introduce additional faults), and these lead to much more complex mathematical relations but still yield an asymptotic approach to zero failure rate (e.g., [SHOO77]).

Several of the more widely used models also remove the assumption of a constant proportionality between fault content and failure rate, thus making k a variable. In particular, it is argued that easy-to-find faults are removed first, and that the faults that remain must therefore, be harder to uncover which means that the value of k decreases as the debugging proceeds (e.g., [GOEL78], [LITT80]). There is some experimental evidence that specific fault types require more runs to be uncovered than other types [NAGE82] and that would support the hypothesis that k decreases with time if the environment remains unchanged.

Most of the models described in the literature use data from software projects that were either in test or were operational, and the parameters were fitted to the data obtained in those environments. However, when the models have been applied to data from other environments, poor results were generally observed [SUKE77], [CURT79], [ANGU83]).

Thus, the objectives of the project have not been attained in past efforts. Yet, prior investigations form a good foundation from which to proceed if the lessons which they represent are thoroughly studied and integrated. The approach of the present project holds great promise that significant improvements in software reliability methodology can be obtained because (a) it combines prediction and estimation techniques over the entire development cycle and (b) it integrates the previously separated efforts in reliability prediction/estimation and software quality metrics.

A third dimension of the research, sponsored by RADC, has been in the area of data collection. The Data and Analysis Center for Software (DACS) is a data repository for software developments with the intent of making that data available for research efforts such as this [GLOS84].

Software quality metrics and software reliability estimation models share a common goal, i.e., predicting or estimating software reliability before the software system is placed into operational use. Information concerning the early prediction of software reliability can be used by software developers in making software engineering decisions in constructing the software and by acquisition managers in making acquisition and resource planning decisions. Part of the motivation for both techniques stems from the accepted concept that the cost of correcting poor reliability is far less expensive early in the life-cycle than during the operational phase.

There are many similarities between metrics and models; both are relatively new, immature techniques that have relied heavily on historical data, not only for development, but also for validation. Despite these similarities, there are also important differences. Historically metrics and models are applied at completely different stages of the development life-cycle; metrics being applicable as early as the requirement phase, and the models only after testing has begun, while the metrics currently do not use that data at all. Models address software reliability alone, while metrics can be used to predict other qualities. Finally, metrics provide data at both the software system and the module level; models generally portray a system perspective. The results of this effort change this situation by combining aspects of metrics and models across the life-cycle.

To adequately address software reliability, both the software "product" and the software development "process" must be considered. In addition, both the "time-dependence" and the "time-independence" aspects of reliability must also be considered. It must also be noted that software reliability can be realized in different forms, depending on the software life-cycle stage. During the software development life-cycle, software quality metrics could be used to derive a Predictive Software Reliability Figure-of-Merit Number, a number calculated from software characteristics or attributes which would make a quantitative statement about future reliability. During Software Performance Testing, System Integration and Testing, and Operational Test and Evaluation (OT&E). A Reliability Estimation Number calculated from test data would represent reliability during those phases. These numbers would serve as indicators or guides to software reliability. During Deployment (or Operation and Maintenance (O&M)), a final reliability assessment would be made on achieved reliability based on actual field data not test data. Instead of an indirect measure of reliability, a

Reliability Assessment Benchmark will involve direct observation of software failures experienced by the system in performing its mission.

1.3 OBJECTIVES OF PROJECT

The objective of this research and development project is the development of a system-oriented methodology that can be used directly for reliability prediction and reliability estimation; first for software, and later for the entire system.

The methodology must provide:

- Guidance for establishing goals/requirements for software reliability at the start of a project.
- Useful measurement of reliability during the early phases of the life-cycle development to permit effective correction of potential faults.
- Guidance for how software reliability numbers could be used for making software engineering decisions across the software development life-cycle.
- A system-oriented view of embedded software.
- A transition bridge from the early life cycle phases of requirements, design, and coding to later phases of operational testing.
- Metrics that evaluate and correlate the quality factors in the requirements and design to the quality factors in the code and test results.

In order to accomplish this goal, it is critical that the technical approach to developing this methodology take into account certain key considerations. Those considerations are:

- The underlying system reliability characterization and prediction technique is oriented toward Software Acquisition Managers, Air Force System Planners, and Program Offices.
- In order for reliability to be built into a system, the above key people must have an early active role in assessing the quality and complexity of system requirements and design, and comparing the estimated or predicted reliability with system requirements and goals.
- The methodology is a result of synthesis and filtering of the many current approaches to reliability prediction and estimation into a system-oriented procedure with a common basis of measurement. A subset of the past research which lends itself to merging the predictive metric techniques

with the reliability estimation models is used.

- Problems which have plagued reliability research in the past and which should be avoided to the degree possible are: poor definitions in term of units of measures; incomplete validation of models; focus on testing/debugging data rather than system structure; in applicability of techniques to early life cycle phases; and quality assurance orientation rather than prediction orientation.

- To reduce data collection and analysis costs, the potential for automating the collection of the measures and using them to produce the Prediction S/W Reliability Figure-of-Merit Number and the Reliability Estimation Number must be considered.

1.4 APPROACH OF PROJECT

Figure 1-1 illustrates the tasks performed during the entire research and development project.

The first task involved establishing a framework. Definitions of the Reliability Figure-of-Merit (prediction) and Reliability Estimation Number (estimation) were also developed. The utility of this approach to Air Force organizations was considered. An interim report documented these findings. The results are described in Section 2 of this report.

The second task involved identifying current measurements that have potential within the framework developed in task one. The approach to using these measurements was developed during that task. The candidate systems for data collection were also identified and preliminary data collection activities, including discussions with practitioners within DoD were initiated. A Phase I final report was documented. The results are documented in Section 3 of this report.

During task three, new measurements were considered for potential utility within the framework. The concentration during this task was in early life-cycle measurements and the development of procedures for calculating the reliability predictors and estimators. An interim report provided the findings to date. These results are also provided in Section 3.

During task four, the methodology was refined by settling on the measurements to be used, determining how the predictive and estimation numbers will be reported and analyzed, and how their impact on system reliability will be analyzed. These results are in Section 5 and 6.

During task five, the measurements were applied to several systems in order to validate their utility. The systems chosen

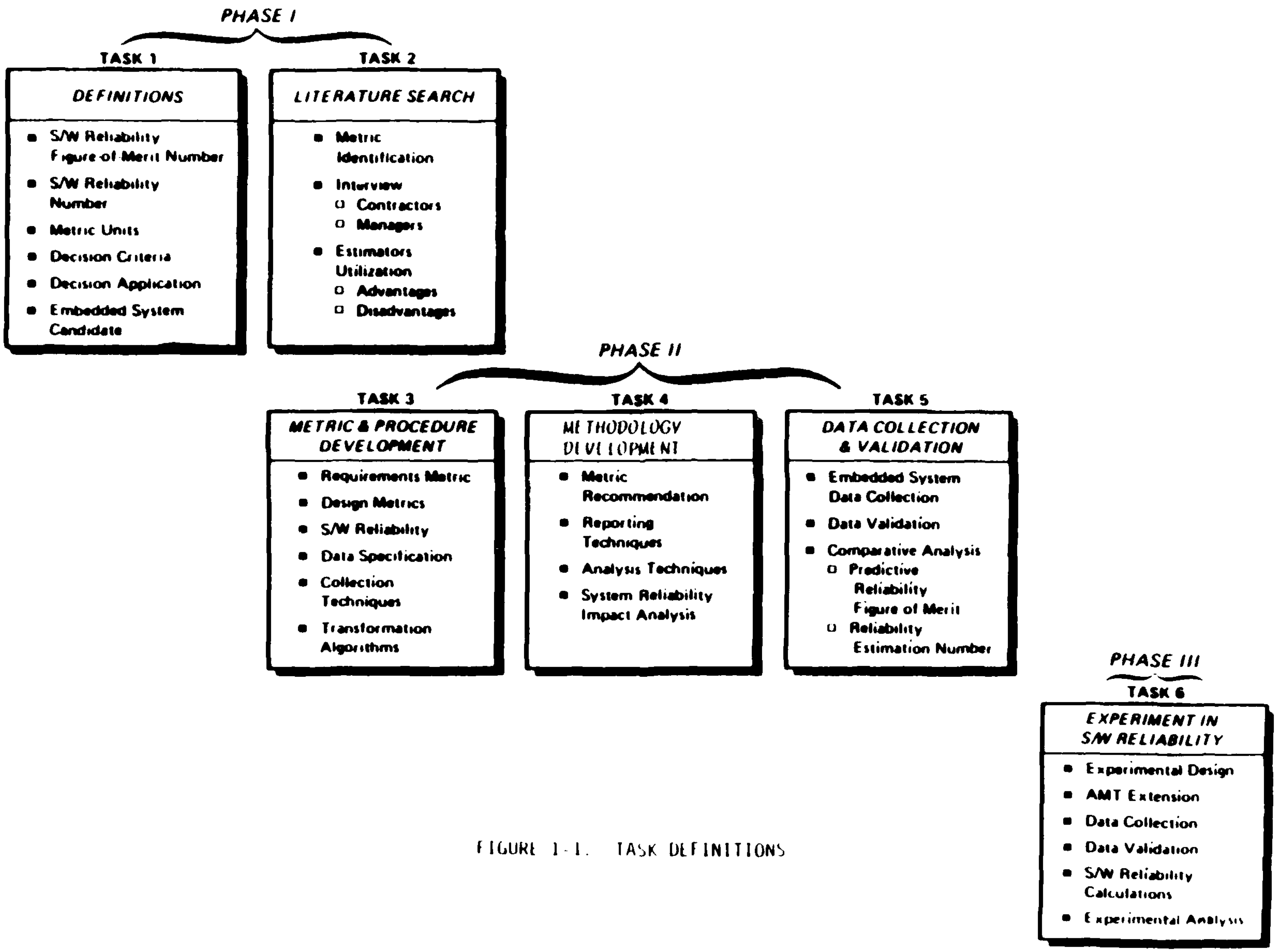


FIGURE 1-1. TASK DEFINITIONS

for data collection in the earlier tasks were used. Statistical analyses of the data collected and the results of the application of the prediction and estimation techniques have been performed. A Phase II Final Report described the results of tasks three, four, and five, which comprised Phase II o* the project. Sections 4 and 5 of this report describe the results of these efforts.

Task six (Phase III) involved an experiment to assess the developed methodology. The methodology was applied in line with a software development and its results assessed. Section 6 of this report describes the findings of this task. An assessment of changes necessary to the AMS was also made during this task. That assessment was documented in another report.

1.5 ORGANIZATION OF REPORT

This report is organized in two volumes. Volume I contains the findings of the project. Volume II contains a Methodology for Predicting and Estimating Software Reliability based on the findings. The methodology is presented in the form of a guide book to aid in its application.

This section provides a brief overview of the sections within this first volume.

Section 1 is the introduction describing the purpose of this report, the objectives of the research effort, some background information, the organization of the report, and an executive summary.

Section 2 describes the framework established in which software reliability measurement will be defined. Definitions and terminology related to this framework are in Appendix A.

Section 3 describes the actual measurements identified during the project. The process we went through to identify the measurements and filter a large initial set to a final set is described.

Section 4 describes the data collected and delivered to RADC as a result of this effort. Further recommendations for data collection and retention are offered.

Section 5 describes the process we went through to demonstrate and validate that these measurements were effective at predicting and estimating reliability. Those measurements that were effective have been retained in the methodology described in Volume II. Those that were not have been either dropped or retained for further investigation/modification.

Section 6 describes the experiment, results, and identifies how the methodology can assist users in taking corrective actions during a software development project.

Section 7 Provides conclusions, recommendations and proposes further research efforts and data collection activities to continue refining the Reliability Prediction and Estimation Methodology. Suggestions for modification of the Sof ware Quality Measurement Framework are also proposed.

1.6 EXECUTIVE SUMMARY

The important results of this effort can be summarized into four areas. Each area is briefly highlighted here with reference to the sections in the report where details can be found

1. Software Reliability Measurements Framework

   A framework is established which spans the life cycle of a software system. The framework acknowledges the inputs of past RADC research in metrics and models as techniques to aid in the prediction and estimation of reliability during the development process. Completing the framework are the specification and assessment aspects of reliability measurement. Within the framework, the specific data needed to measure software reliability and the utility of the measurements to help make sound software engineering decisions is addressed. The framework is presented in Section 2 of this report. Future research and data collection should be focused by this framework.

2. Software Reliability Data

   This research effort probably entailed the most comprehensive data collection/compilation effort attempted to investigate software reliability. Over thirty-three (33) data sources representing 59 systems and over 5 million lines of code were accessed (including the RADC Data and Analysis Center for Software and the NASA Software Engineering Laboratory Data Base). Because of the diversity of the data collected, more generally applicable observations about software reliability could be made. This extensive data base supported the development of the preliminary guidebook for making reliablility predictions and estimations. Summary data and examples of detailed data collected are presented in section 4 of this report.

3. Preliminary Guidebook for Software Reliability Prediction and Estimation

   A guidebook (Volume II of this report) was developed to allow software reliability engineers to practice the techniques developed during this research effort. Utilizing the data collected and the metrics derived from analysis, procedures are provided which allow predictions and estimations to be made at various milestones during a software development project.

4. Experiment Demonstrating Prediction and Estimation Techniques

Section 6 of this report describes the application of the Guidebook to an actual project. Comparsions of the predictions and estimations with actual results are provided.

2.0 A FRAMEWORK FOR SOFTWARE RELIABILITY PREDICTION AND ESTIMATION

2.1 THE FRAMEWORK

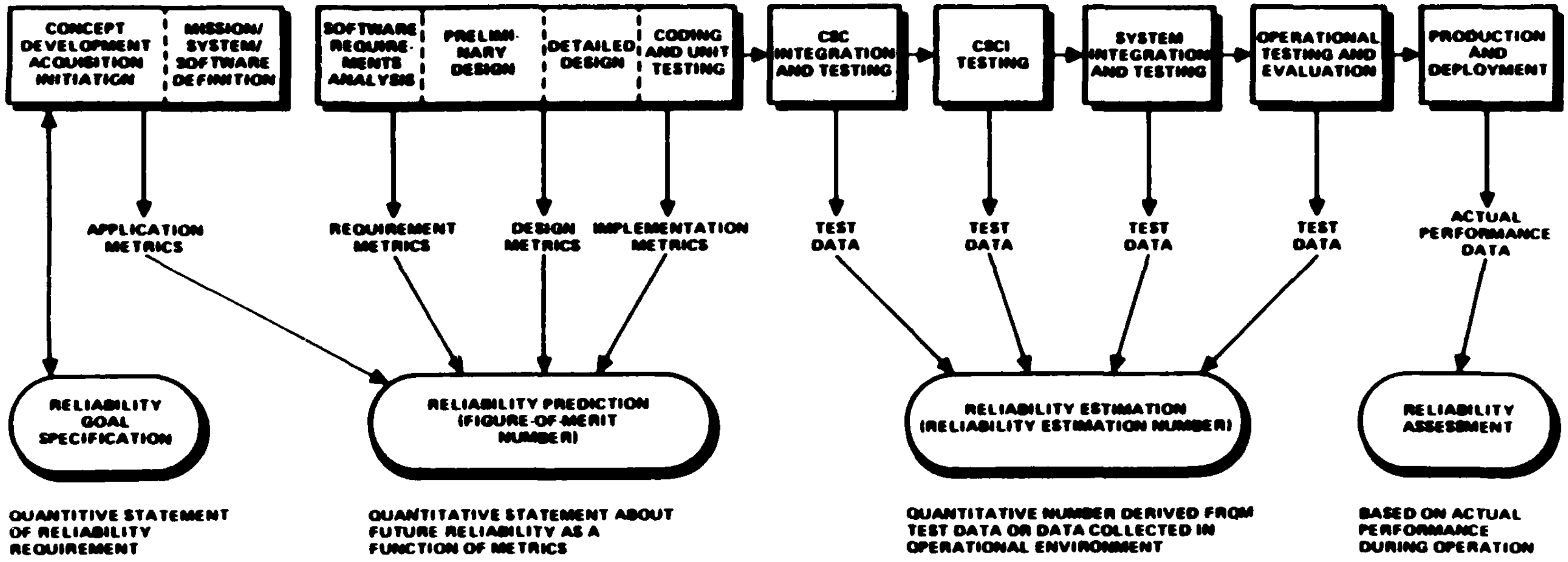
The current technology in software reliability, as a result of past research efforts, has been, for the most part, not accepted by the reliability practioners. On one hand, models of software reliability using metrics related to structural characteristics of the software provided predictions of the number of faults expected in a portion of the code. This had little relevance to reliability engineers because their orientation is time (e.g., failure rate or MTBF). On the other hand, models of software reliability using failure detection rates during testing provides relevant data, but because of necessary model assumptions, the lateness in application, and the sensitivity to the testing approach, the models also did not meet practioner's needs.

A framework developed during Phase I of this effort attempts to build upon both approaches and span the entire life-cycle in applicability. Figure 2-1 illustrates the Reliability Measurement Framework.

The framework illustrates the following important characteristics:

- The framework illustrates reliability measurement as a life cycle activity.
- The framework includes specification of reliability goals, prediction of reliability during the early phases of development, estimation of reliability during the later phases of development, and assessment of the achieved reliability during operations and maintenance (deployment).
- The framework combines the measurement techniques of software quality metrics and reliability models.
- The techniques are described in units which are consistent.
- The measurement techniques are also described in terms consistent with actual reliability measurement.
- The approach taken will lend itself to combination with traditional hardware reliability concepts so system reliability can be addressed.

During the concept development phase, a technique to specify the software reliability goal of the system is needed which will be compatible with similar hardware reliability goals. The predic-


CONCEPT DEVELOPMENT ACQUISITION INITIATION
MISSION/ SYSTEM/ SOFTWARE DEFINITION
SOFTWARE REQUIRE-MENTS ANALYSIS
PRELIM-INARY DESIGN
DETAILED DESIGN
CODING AND UNIT TESTING
CSC INTEGRATION AND TESTING
CSCI TESTING
SYSTEM INTEGRATION AND TESTING
OPERATIONAL TESTING AND EVALUATION
PRODUCTION AND DEPLOYMENT
APPLICATION METRICS
REQUIREMENT METRICS
DESIGN METRICS
IMPLEMENTATION METRICS
TEST DATA
TEST DATA
TEST DATA
TEST DATA
ACTUAL PERFORMANCE DATA
RELIABILITY GOAL SPECIFICATION
RELIABILITY PREDICTION (FIGURE-OF-MERIT NUMBER)
RELIABILITY ESTIMATION (RELIABILITY ESTIMATION NUMBER)
RELIABILITY ASSESSMENT
QUANTITIVE STATEMENT OF RELIABILITY REQUIREMENT
QUANTITATIVE STATEMENT ABOUT FUTURE RELIABILITY AS A FUNCTION OF METRICS
QUANTITATIVE NUMBER DERIVED FROM TEST DATA OR DATA COLLECTED IN OPERATIONAL ENVIRONMENT
BASED ON ACTUAL PERFORMANCE DURING OPERATION

Note. THE RELIABILITY FIGURE OF MERIT NUMBER AND THE RELIABILITY ESTIMATION NUMBERS MUST BE DEFINED IN THE SAME UNITS OF MEASUREMENT SO THAT THEY MAY BE MEANINGFULLY COMPARED AND RELATED TO EACH OTHER

FIGURE 2-1. FRAMEWORK FOR SOFTWARE RELIABILITY

tion technique (Reliability Figure-of-Merit) is based on metrics (quantitative measures) that can be taken during early phases of development. These metrics are predictive or indicative in nature. They are based on structure, development techniques and methods, and environment. The estimation technique (Reliability Estimation Number) is based on test results. The Estimation Number is refined as testing progresses. During operation and maintenance, reliability assessment is conducted. This assessment consists of observing the actual achieved reliability and describing it quantitatively.

This last aspect of the framework is very important to the useability of the methodology. By requiring that the techniques relate to actual measurement, the likelihood of acceptance with the practitioner community is much greater. The techniques become more understandable and relate to goals that are specified.

To make the approaches compatible, software reliability must be expressed in terms of failure rate. The time unit of measure of the failure rate must be in terms of execution time because this is conceptually equivalent to hardware operating time. Figure 2-2 illustrates this relationship between hardware and software reliability. Appendix A provides definitions and terminology related to this framework.

2.2 UTILITY OF RELIABILITY MEASUREMENT TECHNIQUES

A major goal of this study is to define reliability prediction and estimation concepts so they are useful to Air Force users. A first step in achieving this goal is to identify what needs these concepts must satisfy, or what utility they can provide to Air Force users.

The Air Force organizations to be discussed are end-users (e.g., SAC and TAC), System Acquisition Managers (SAMs) and System Program Offices (SPOs) such as ESD and ASD, Air Force Plant Representatives (AFPRO), Test and Evaluation organizations such as AFOTEC, Life Cycle Agents such as ALCs (AFLC), research organizations such as RADC, developers (in most cases contractors), and Independent Verification and Validation contractors. Figure 2-3 illustrates the relationship of these organizations on a typical development.

The techniques these organizations will be involved in using include specifying reliability goals, predicting reliability during early phases of the development, estimating reliability during the testing phases, observing actual reliability performance (assessment) during operations and maintenance, and assessing what improvements can be initiated to improve the design and production process to improve software reliability.

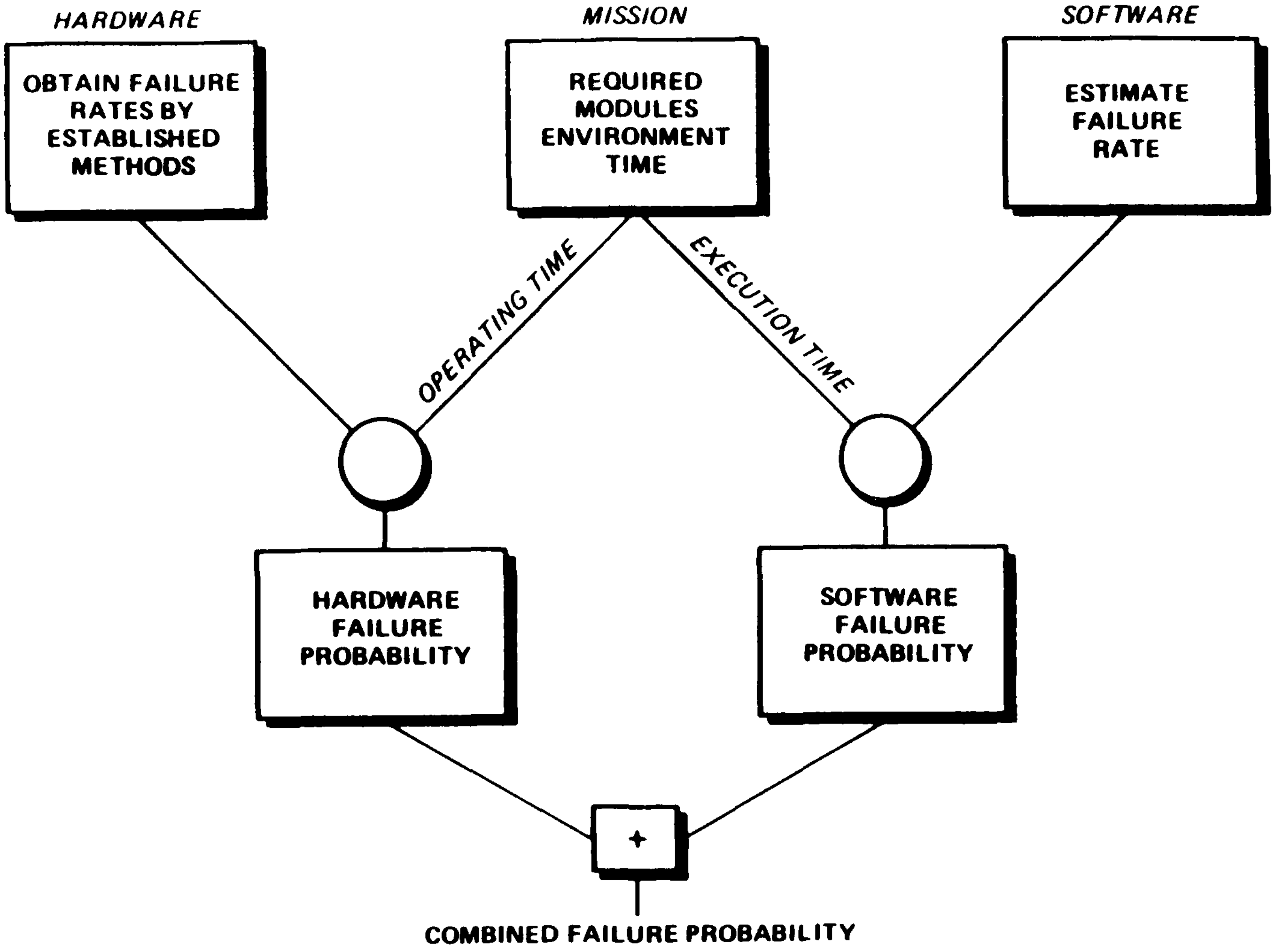
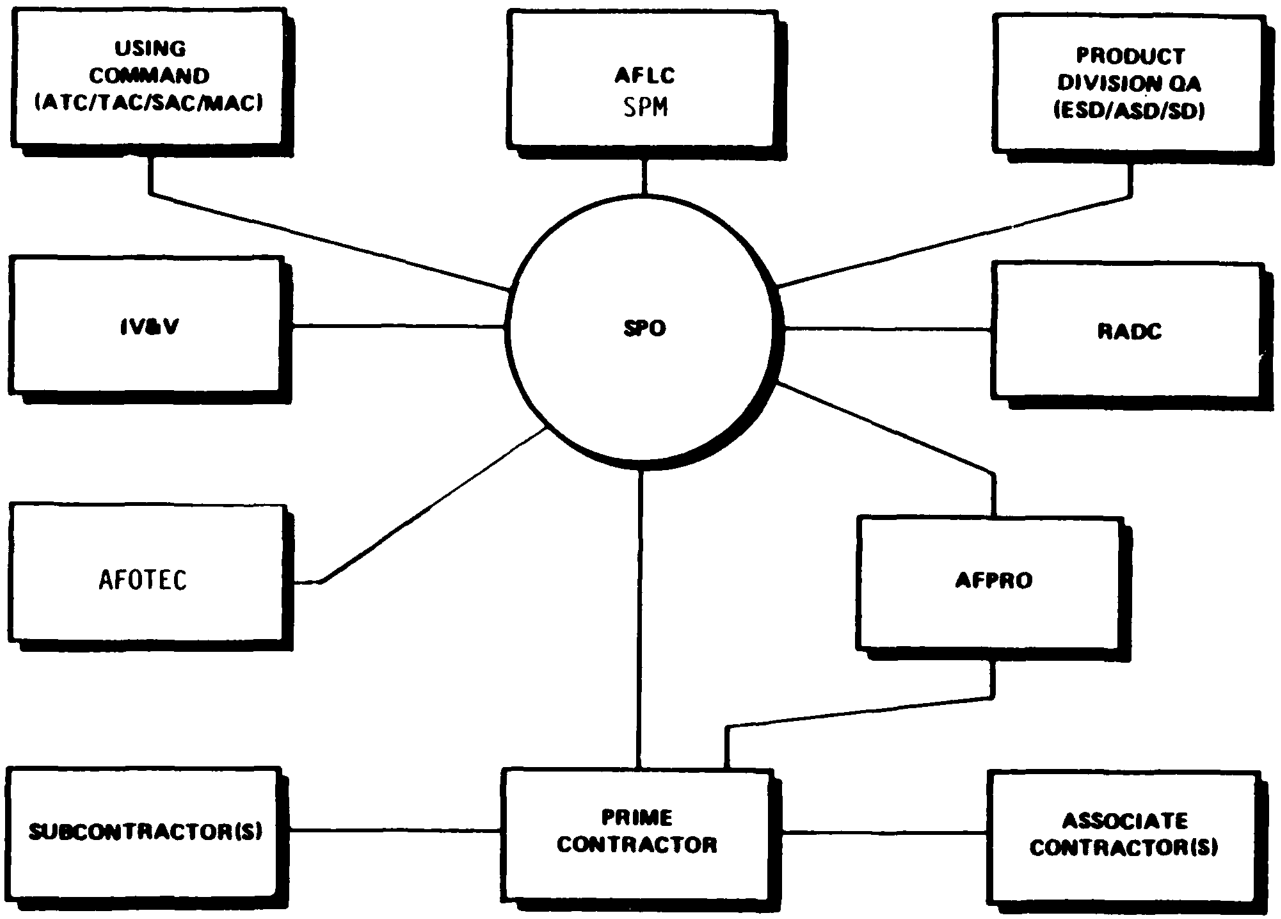

HARDWARE
MISSION
SOFTWARE
OBTAIN FAILURE RATES BY ESTABLISHED METHODS
REQUIRED MODULES ENVIRONMENT TIME
ESTIMATE FAILURE RATE
OPERATING TIME
EXECUTION TIME
HARDWARE FAILURE PROBABILITY
SOFTWARE FAILURE PROBABILITY
+
COMBINED FAILURE PROBABILITY

FIGURE 2-2. RELATIONSHIP BETWEEN HARDWARE AND SOFTWARE RELIABILITY



LEGEND:

ATC - Air Training Command
TAC - Tactical Air Command
SAC - Strategic Air Command
MAC - Military Air Command
IV&V - Independent Validation and Verification

AF - Air Force
AFLC - AF Logistics Command
AFPRO - AF Plant Representative Office
SPO - System Program Office
RADC - Rome Air Development Center

ESD - Electronics System Division
ASD - Aeronautical System Division
SD - Space Division
QA - Quality Assurance
SPM = System Program Manager
AFOTEC = AF Operational Test and Evaluation Center

FIGURE 2-3. AIR FORCE ACQUISITION RELATIONSHIPS INVOLVED IN QUALITY METRICS FUNCTIONS

Their use of the four techniques and their involvement in the various phases of a development is illustrated in Figure 2-4. The following paragraphs describe the involvement in more detail.

2.2.1 Utility During Concept Development/Acquisition Initiation/Mission and System Requirements Definition Of A Major Project

During the concept development of a major project that is dependent on software for a critical part of its function, there is frequently a general concern about the ultimate reliability that can be attained. The end users and SAMs are involved in this phase. Reliability may be required in connection with safety, as in a digital fly-by-wire system for aircraft, or it may be desired on the basis of general mission goals, as in an area air defense system. The central question in both circumstances is "will the operational reliability meet the minimum requirements for the intended application?" If this is answered in the affirmative, the project may proceed. If it is answered in the negative, alternative approaches will have to be investigated. Thus at concept development, a predicted reliability number is needed for the concept architecture proposed to compare it with the required system reliability. Required reliability must be specified as a goal and incorporated in system requirements specifications and acquisition documents.

If the forecasted reliability satisfies the minimum requirements (and if other conditions are met), the project acquisition will be initiated. Here the concern shifts to establishing milestones at which it can be determined whether adequate progress is being made toward meeting the reliability goals. Thus, there is at least an implicit requirement for a model of the process by which reliability is being attained, such as the elimination of faults in the design and code. Three related questions sum up the primary objectives for this phase:

- "What milestones can be established to verify the attainment of reliability goals during the course of the development?",

- "What are the key measures that can be obtained at each one of the milestones?", and

- "What techniques should be required of the developer to promote reliable software development?".

These questions demand a detailed understanding of the software failure process. The answers to these questions result in a software reliability test plan, at least to the level where tests are identified by name, scope of the system under test, and test objectives. The System Program Office (SPO), the developer, and the Test Agent are involved in this process of identifying definitive reliability goals and test plans.

	TECHNIQUES				*LIFE CYCLE PHASE ACTIVITIES*								
	GOAL SPECIFICATION	PREDICTION	ESTIMATION	ASSESSMENT	CONCEPT DEVELOPMENT/ ACQUISITION INITIATION	MISSION/SYSTEM/SOFTWARE REQUIREMENTS	PRELIMINARY & DETAILED DESIGN	CODING AND UNIT TESTING	CSC INTEGRATION AND TESTING	CSCI-LEVEL TESTING	SYSTEM INTEGRATION AND TESTING	OPERATIONAL TEST AND EVALUATION	PRODUCTION AND DEPLOYMENT
END-USER	●			●	●							●	●
SAM/SPO	●	●	●		●	●	●	●	●	●	●	●	
AFPRO		●	●		●		●	●	●	●	●	●	
AFOTEC			●							●		●	
LIFE CYCLE AGENT (ALC)		●	●	●								●	●
RESEARCH AGENT (RADC)	●	●	●	●									
DEVELOPER (CONTRACTOR)	●	●	●			●	●	●	●	●	●	●	●
IV&V (CONTRACTOR)		●	●	●			●	●	●	●	●	●	●

FIGURE 2-4. AIR FORCE ORGANIZATIONAL INVOLVEMENT IN RELIABILITY MEASUREMENT

2.2.2 Utility During Early Software Development Phases of Requirements Analysis, Preliminary Design Detailed Design and Coding

During the phases of system development, the SAM/SPO management is concerned with trade-offs of broad scope, e.g., allocation of functions to hardware, software, and personnel. The principal reliability concern in these activities is the effect of the decisions on the global reliability of the system, and a single measure of forecasted software reliability in the operational environment is usually sufficient. These objectives are similar to those described under the planning phase above.

As the development proceeds through the development milestones, the software reliability goals that were established during the initiation phase should be evaluated and technical management will want to determine that the milestones have been attained. This may involve direct measurement of software reliability or, particularly at the early milestones, evaluation of predictors of software reliability. At this stage the establishment of objective and accessible measurement criteria is essential.

If it is determined that milestone objectives have not been attained, a recovery plan must be prepared. Typically, this involves corrective actions modifying the software system architecture, the design, or the code.

Software Development Management is interpreted here as those organizational activities in a project that are directly charged with oversight of the software development, test, and integration. The objectives of the higher level managers of the software activities within the developing organization are expected to have similar objectives, particularly where software development is subcontracted and must be managed as a separate activity.

In the context described above, software management has received operational reliability goals and requirements to be met at specified milestones during the development which were generated as outlined in the preceding paragraphs. These goals must be allocated to individual software segments, and it is also generally desired to establish more detailed evaluation criteria so that the probability of attaining the milestone requirements can be gauged during the development process. From these responsibilites arise objectives for software reliability forecasting at a much more detailed level than found in the prior discussion. At the same time, software management has access to much more specific information about the structure, content, and development environment of the product.

Where the attainment of milestones or of the ultimate reliability goals appears in doubt, means of gauging the effects of several alternatives for reliability improvement are desired. Candidate alternatives may involve a new design for the program or for the data structure, improved test techniques, or the adoption of software fault containment or fault tolerance techniques. These types of software engineering decisions will be driven by the reliability predictors. The reliability prediction and estimation techniques should support an objective and accurate evaluation of the effects of these alternatives. During this phase, the forecasting techniques are used to evaluate progress and assist in the reliability engineering. A quality assurance or reliability engineering group within the developer's organization or an IV&V contractor would most likely be involved in taking these detailed measures. The software development team within the developer's organization would use measures to make software engineering decisions.

2.2.3 Utility During Test Phases and Acceptance

The observed system reliability during the various phases of testing and eventually during acceptance testing can be the basis for acceptance/rejection of the system. If a goal is contractually stated and the acceptance test procedure specifically identifies that goal as an acceptance/rejection criterion, then use of this technique can have significant importance to the developer. The developer is involved in performing system testing. An independent Test and Evaluation organization or an IV&V contractor may be involved in conducting independent tests to assess reliability. The SPO and SAM are involved in accepting the system. The Test Agent is involved in operational testing phases.

2.2.4 Utility During Transition To Operational Use (Deployment) and Operations and Maintenance

Although the planning and initiation activities had generated a time phased series of milestones that should lead to the desired software reliability in operational use, there usually arise a considerable number of questions about software reliability as the date for cut-in approaches. The goals established during planning were of necessity quite general and may no longer be applicable to the structure of the system and software as they are being delivered. It is quite typical to observe during the cut-in period many failures associated with the software that are not truly software failures but are the result of procedural mistakes or of inconsistencies between the specified and the actual environment. The objectives of software reliability at this point relate primarily to reporting and measurement procedures, with emphasis on distinguishing between events where the software failed to meet its specification (the frequency of these can be interpreted as indicative of operational reliability) and events that are primarily due to the transition process and which are therefore not expected to persist during steady state operation. The life-cycle agent and end user are involved in this process.

After a system has become operational, a software reliability goal is to exhibit a pattern of continued decrease of failure frequency and, concomitant with this, to identify and prevent causes of increasing failure frequency. The utility of the reliability measurements are the ability to assess the reliability actually achieved within the system. Typical causes of poor reliability include inadequate software maintenance, instability of the hardware or software configuration, and lack of communication regarding changes in user requirements or expectations. The emphasis is on measurements that are efficient in identifying changes in trends. Again the end user and life-cycle agent play key roles in maintaining and improving the reliability performance of the system.

2.3 SOFTWARE RELIABILITY ENGINEERING MANAGEMENT

Figure 2-5 identifies many of the activities sited in the above paragraphs according to detailed life-cycle phases. The availability of specific measurements and predictive and estimation techniques will facilitate the performance of these activities during software developments. These activities represent are Software Reliability discipline that should be incorporated in software development. This discipline has aspects that are management-related, development-related, quality assurance-related, and test-related.

Figure 2-6 highlights the types of questions that the reliability measurement techniques will help answer.

CONCEPT DEVELOPMENT/ ACQUISITION INITIATION	MISSION/ SYSTEM REQUIREMENTS	SOFTWARE REQUIREMENTS	PRELIMINARY AND DETAILED SOFTWARE DESIGN	CODING AND UNIT TESTING	CSC INTEG AND TEST/ CSCI-LEVEL TESTING	SYSTEM INTEGRATION AND TESTING/ OT&E	OPERATIONS AND MAINTENANCE
■ Establish Reliability Requirements ■ Perform High Level Tradeoffs ■ Relate Reliability To User	■ Set Reliability Goals For System ■ Allocate Reliability Goals To Hardware and Software ■ System Reliability Assessment (SDR)	■ Allocate Reliability Requirements To Software ■ Analyze Testability of Requirement ■ Analyze Feasibility of Requirements ■ SRR	■ Decompose and Budget Reliability Requirements To Software Components ■ Establish Design Practices To Encourage Reliable Software Design ■ Analyze/ Simulate Reliability Performance ■ Predict Software Reliability ■ PDR ■ CDR	■ Build In Reliability ■ Establish Coding Standards To Encourage Reliable Software Production ■ Conduct Unit Testing/ Debugging To Remove Module Level Faults ■ Prototype Builds For User Feedback ■ Predict Software Reliability	■ Test To Requirements ■ Test Thoroughness Evaluation ■ Maintain Standards During Rework ■ Insure Test Quality ■ Regression Testing ■ Estimate Software Reliability ■ Problem Report Statistics ■ TRR ■ Acceptance Testing	■ Hardware/ Software Error Analysis ■ Hardware/ Software Reliability Integration ■ System Test Thoroughness Evaluation ■ Insure Test Quality ■ Regression Testing ■ Estimate System Reliability ■ Test Assessment In Operational Environment	■ Regression Testing ■ Quality Assurance ■ Reliability Measurement

FIGURE 2-5. SOFTWARE RELIABILITY ENGINEERING MANAGEMENT

Users of Reliability Measurement Concepts	Life Cycle Phases									
	Acquisition Initiation/ Concept Development/ Mission/System Requirements Definition	Software Development						Systems Integration & Testing	Operational Test and Evaluation	Operations and Maintenance
		Software Requirements Analysis	Preliminary Design	Detailed Design	Coding & Unit Testing	CSC Integration & Test	CSCI Level Testing			
End User	• What Reliability do I need for my application?								• Does system perform as expected in operational environment?	• How is the system performing now that I'm using it?
Acquisition Manager/ System Program Office	• What Reliability is needed? • How can I specify that Reliability as a contractual goal? • What are the cost/budget impacts of different levels of Reliability? • What is predicted Reliability of different high level system architectural alternatives being considered?	• What milestones should be established to help assess progress toward Reliability goals? • What measures can be taken to evaluate the progress toward Reliability goals? • What standards should be required to guide the developer toward meeting Reliability goals?					• How can SAM/SPO evaluate whether software should be accepted? • Should software be accepted?	• How can SAM/SPO evaluate whether system should be accepted? • Should system be accepted?		
Developer	• What Reliability goals have been established?	• How do the Reliability goals flowdown to software elements? • How can Development Management monitor progress toward Reliability goals? • How can Reliability be built into software?			• How can progress toward Reliability goals be demonstrated at interim milestones? • If problems are identified, what software engineering corrective actions must be taken to correct deficiency?		• How can software Reliability be demonstrated for acceptability?	• How can system Reliability be demonstrated for acceptability?		
IV&V QA	• What are Reliability goals of system?	• How have they been allocated to the hardware and software elements? • Does the software architecture support attainment of these Reliability goals?			• Does the development process support attainment of the Reliability goals? • Are standards being complied with? • Do the predictive Reliability measures indicate achievement of Reliability goals?		• Does software perform reliably during acceptance testing? • Does observed software Reliability during testing indicate achievement of Reliability goals?	• Does system perform reliably during System Integ and Test? • Are failures, hardware or software? • What trends are illustrated by failure types/rates?		• Is operational Reliability sufficient to perform mission?
Test Agent							• Does observed Reliability during testing indicate achievement of Reliability goals? • Is software maintainable or will lack of maintainability contribute to operational performance deficiencies?	• Does observed Reliability during testing indicate achievement of Reliability goals? • Is software maintainable or will lack of maintainability contribute to operational performance deficiencies?	• Does observed system Reliability indicate achievement of Reliability goals? • Is system maintainable or will lack of maintainability contribute to operational performance deficiencies?	• Do modifications maintain Reliability performance?
Life Cycle Agent	• What are support requirements to maintain system with stated Reliability goals?				• What Reliability problems were reported during development? • Which problems were fixed and which were not?		• What software Reliability problems were reported during testing? • Which problems were fixed and which were not? • What tests were run to evaluate Reliability?	• What Reliability problems were reported during System Integ and Test? • Which were fixed, which were not? • What tests were run to evaluate Reliability?	• What Reliability problems were reported during OT&E? • Which were fixed, which were not?	• How can Reliability be maintained during modification activities? • How can Reliability be improved if it is not acceptable? • What measures can be applied to assess Reliability during modification?

FIGURE 2-6 QUESTIONS TO BE ANSWERED

3.0 CANDIDATE RELIABILITY MEASUREMENTS

The Software Reliability Measurement Framework illustrated in Figure 2-1 in Section 2, identified two measurement objectives that were the focus of this research effort. They are a Predictive Software Reliability Figure-of-Merit (RP) and a Reliability Estimation Number (RE). The predictive RP is derived from measurements taken in the early life cycle phases of a development, when based on the characteristics of the evolving software system a prediction can be made of the reliability of the software. The RE is an estimation of the reliability based on the observed failure rate of the software during the test phases of the development. This section describes the candidate measurements which were identified for each of those numbers. Also described in this section are the relationship of these candidate metrics to the RADC Software Quality Measurement Framework, when during the life-cycle these candidate measurements apply, and Data Collection Procedures for calculating the metrics. Section 4 of this report describes the data collected to calculate these metrics. Section 5 describes the process and results of the validation efforts with these metrics.

3.1 SOFTWARE QUALITY MEASUREMENT FRAMEWORK

A Software Quality Measurement Framework was established in Factors in Software Quality, RADC-TR-77-369. That framework had a basic structure illustrated in Figure 3-1. From that initial report, four quality factors are identified that relate and impact software and system reliability:

Software Reliability: The extent to which a program can be expected to perform its intended function with required precision.

Software Correctness: The extent to which a program satisfies its specifications and fulfills the user's mission objectives.

Software Maintainability: The effort required to fix an error in an operational program.

Software Testability: The effort required to verify the specified software operation and performance.

A more recent report, Specification of Software Quality Attributes, RADC-TR-85-37, expands these factors to the following:

Reliability: Extent to which the software will perform without any failures within a specified time period.

Survivability: Extent to which software will perform and

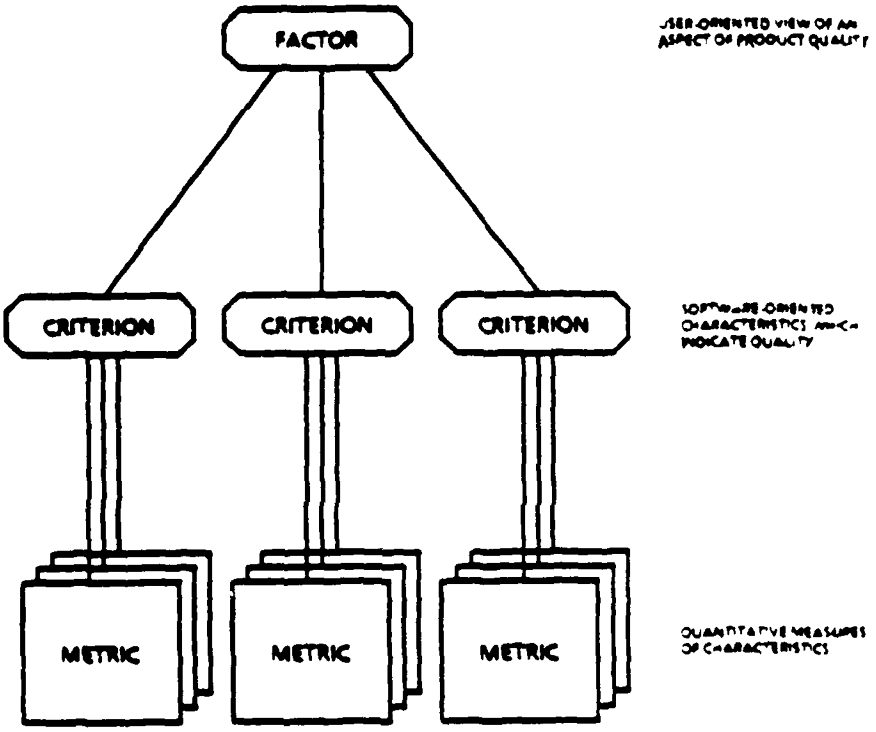

FACTOR
USER-ORIENTED VIEW OF AN ASPECT OF PRODUCT QUALITY
CRITERION
CRITERION
CRITERION
SOFTWARE-ORIENTED CHARACTERISTICS WHICH INDICATE QUALITY
METRIC
METRIC
METRIC
QUANTITATIVE MEASURES OF CHARACTERISTICS

FIGURE 3-1. SOFTWARE QUALITY MEASUREMENT FRAMEWORK

support critical functions without failures within a specified time period when a portion of the system is inoperable.

Correctness: Extent to which the software conforms to its specifications and requirements.

Maintainability: Ease of effort for locating and fixing a software failure within a specified time period.

Verifiability: Relative effort to verify the specified software operation and performance.

Table 3-1 illustrates the criteria and metrics related to these factors. Each of these metrics were considered in arriving at the candidate measurements for the RP and RE. Also considered specifically for applicability to the RE were the reliability models mentioned in Section 1 and described in [GOEL83].

3.2 A SOFTWARE RELIABILITY MEASUREMENT MODEL

The framework presented in Section 2 represents a life-cycle view of software reliability measurement. The heart of the framework is the ability during the development phases to predict and estimate software reliability. These predictions and estimations are comparable to the specified reliability requirements and eventually to the observed operational reliability.

3.2.1 A Model Of The Software Failure Process

In order to identify the software measurements to be used to predict and estimate software reliability we need to understand how software fails (i.e., what we are predicting and estimating) and how we can organize the candidate measures according to their value as predictive or estimation metrics.

Software does not fail in the sense of a permanent physical state change such as is usually associated with hardware failures. Nevertheless, it has become customary to refer to software failures as a shorthand term for failures in the computing process which are caused by the software. A graphical representation of that failure process is shown in Figure 3-2. In the strictest sense, the failure is an event that causes a binary bit pattern inside the computer to take a wrong value, shown inside the larger box in the figure.

Typically, this event is not actually observed, but the evidence that a failure has occurred is found in an incorrect value at the output of the computer, i.e., an error (as defined in Appendix A). Not every error is observed, and since the reliability values produced by the prediction and estimation techniques should agree with those eventually observed, the predictions and estimations must be adjusted for the degree to which errors are expected to be observed. The observation takes place in the

TABLE 3-1. CANDIDATE METRICS FROM SOFTWARE QUALITY MEASUREMENT FRAMEWORK

FACTOR	*CRITERION*	*METRIC ACRONYM*	*METRIC*
R	ACCURACY	AM.1	ACCURACY CHECKLIST
R,S	ANOMALY MANAGEMENT	AM.1	ERROR TOLERANCE/CONTROL
		.2	IMPROPER INPUT DATA
		.3	COMPUTATIONAL FAILURES
		.4	HARDWARE FAULTS
		.5	DEVICE ERRORS
		.6	COMMUNICATIONS ERRORS
		.7	NODE/COMMUNICATIONS FAILURES
R,M,V	SIMPLICITY	SI.1	DESIGN STRUCTURE
		.2	STRUCTURED LANGUAGE OR PREPROCESSOR
		.3	DATA AND CONTROL FLOW COMPLEXITY
		.4	CODING SIMPLICITY
		.5	SPECIFICITY
		.6	HALSTEAD'S LEVEL OF DIFFICULTY
S	AUTONOMY	AU.1	INTERFACE COMPLEXITY
		.2	SELF SUFFICIENCY
S	DISTRIBUTEDNESS	DI.1	DESIGN STRUCTURE
S,M,V	MODULARITY	MO.1	MODULAR IMPLEMENTATION
		MO.2	MODULAR DESIGN
S	RECONFIGURABILITY	RE.1	RESTRUCTURE
C	COMPLETENESS	CP.1	COMPLETENESS CHECKLIST
C,M	CONSISTENCY	CS.1	PROCEDURE CONSISTENCY
		CS.2	DATA CONSISTENCY
C	TRACEABILITY	TC.1	CROSS REFERENCE
M,V	SELF DESCRIPTIVENESS	SD.1	QUANTITY OF COMMENTS
		.2	EFFECTIVENESS OF COMMENTS
		3	DESCRIPTIVENESS OF LANGUAGE
M,V	VISIBILITY	VS.1	UNIT TESTING
		.2	INTEGRATION TESTING
		.3	CSCI TESTING

LEGEND:

R = RELIABILITY	M = MAINTAINABILITY
S = SURVIVABILITY	V = VERIFIABILITY
C = CORRECTNESS	

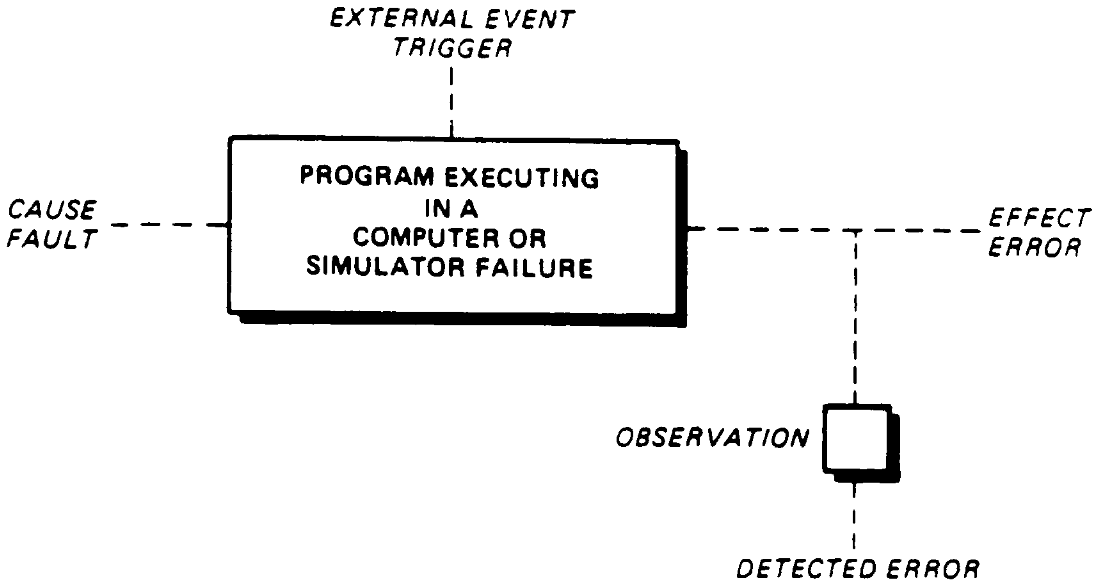


FIGURE 3-2. BASIC SOFTWARE FAILURE MODEL

operating environment, and the methodology for accounting for observation in the estimation is part of an environment factor.

Some faults in the code will produce an error during every execution. These are normally corrected very early during checkout by the developer even before the program enters formal testing. Failures that are of concern in software reliability measurement for Air Force projects usually come about when a rare external event (data set or computer state) causes the execution of the code to differ in some way from the routine manner. A software fault that had previously been present, but not resulted in an error has thereby been revealed. Both the presence of faults in the code and the occurrence of triggering events will, therefore, affect software reliability.

3.2.2 Organization Of Software Reliability Measurements

Two broad classes of software reliability metrics have been addressed in the literature, based, respectively, on fault content of the code and on the number of failures encountered during service. The common normalized forms of these are fault density and failure rate. Because the latter measure can be combined with conventional hardware reliability metrics to yield a single expression for computer system reliability it is being given preference. However, there are some situations in which fault density is either the only measure available or is a more convenient expression to use. Therefore, it is also covered in the following discussion.

3.2.2.1 Fault Density

The software user wishes to procure fault-free code, and the software developer has economic incentives to want to meet the user's requirements. It is recognized that completely fault-free code for a large project is not within the present capabilities, and thus a measure for relative freedom from faults is required. Fault density has been found a useful and meaningful metric. One of the first to provide quantitative data on fault density was F. Akiyama [AKIY71]. He reported an average fault density of 1% in programs entering formal test, and this number has been repeatedly confirmed in other publications. Modern programming techniques have produced some improvement, and a declining trend has been noted. For recent HOL programs, an order of magnitude improvement, .1%, appears to be representative [HECH83].

Fault density can be expressed as the number of faults found in total lines of code or in executable lines of code, and a distinction must be made between these. The measure used in this report is based on executable lines. It is also important to recognize that a single line of HOL code usually replaces 2 to 8 lines of assembly language code, depending on the higher-order language.

Fault density has the following advantages as a reliability metric:

- It appears to be a fairly invariant number.
- It can be obtained from commonly available data.
- It is not directly affected by variables in the environment (but testing in a stressful environment may produce a higher value than testing in a passive environment).
- Conversion among fault density metrics is fairly straightforward (see above).
- The metric facilitates combination of faults found by inspection with those found during execution since the time element of the later is not accounted for.

The major disadvantages are:

- It cannot be combined with hardware reliability metrics.
- It does not relate to observations in the user environment.
- There is no assurance that all faults have been found.

3.2.2.2 Failure Rate

The incidence of software failures (as distinct from the presence of faults in the code) is viewed as an undesirable characteristic by the user. The frequency of failures in a specified time interval is therefore, a measure of unreliability as seen by the user, or, conversely, the time between failures is a measure of reliability. Metrics of this type based on elapsed time (also referred to as wall clock time) are not meaningful for assessment of the inherent reliability of the software product because they are not directly related to the exposure to failure. Thus, for a computer that is not in use during weekends it will be found that the software failure rate (in wall clock time) during that period is a very satisfactory zero. Unfortunately, during the week when it is in use, it has a finite value. This has given rise to some very erroneous assessments of software reliability because the elapsed time failure rate tends to increase during periods of heavy test activity simply because more usage hours are being logged per calendar day. The increasing trend causes concern, reflected in yet higher test activity and higher apparent failure rates.

To avoid these inconsistencies, failure rates based on execution time have been proposed, and their use has led to much more satisfactory results [MUSA75, HECH77]. Failure rates based on execution time or an alternative, computer operation time, will be used throughout this project. Execution time is the interval during which the central processing unit (CPU) of the computer executes the program. It is only during execution of the program that failures will be encountered. The ratio of execution time to wall clock time may, therefore, be thought of as the duty cycle of the software.

On most mainframes, the operating system reports the execution time for each program or project on a run basis and also computes daily, weekly, or monthly totals. Where these reports are not available, execution time may be expressed in computer operation time, the time during which the computer (as contrasted with the CPU) executes the program. Computer operation time exceeds CPU time (in the range of two to ten times CPU time) because it also includes time for mass storage access, output functions, etc. Proper methods of converting computer time to CPU time or equivalent acceptable measures are discussed later in this section.

Failure rate measurements based on execution time have the following advantages:

- Observable and meaningful in the operating environment.
- Can be computed over any time interval limited only by statistical averaging considerations.
- Can with proper procedures be combined with hardware failure rate to yield a computer system failure rate.

They have the following disadvantages:

- Affected by conditions in the environment.
- Do not include faults found by inspection.
- Require measurement or estimation of execution time.

It is intuitive that fault density is a self-normalization metric, i.e., it measures a characteristic of the code that is not directly affected by the length of the program. The execution-time-based failure rate is self-normalizing in the same manner because a long program will have a longer running time than a short one.

3.2.2.2.1 Execution Ratio

There are some environments in which it is possible to obtain the computer time but not the execution time, e.g., avionics computers and militarized microcomputers. Failure rate measurements based on computer time can also be used for monitoring the relative progress of a given software package in the same manner as the failure ratio discussed in the subsequent paragraph. These failure rate measurements can also be used for comparisons between modules as long as all run on the same computer type. Failure rate estimation based on computer time can be implemented in this manner.

However, there will be many instances in which it is desirable to convert computer time to execution time, particularly in the utilization of software reliability prediction. A number of methods can be used for this conversion:

- Running a benchmark HOL program on a mainframe on which execution time will be reported, and then running the same test case on the target computer.
- Running a program on the target computer in a manner that will eliminate or minimize disk access (e.g., by putting data in memory) and output operations, thus obtaining essentially an execution time measurement, and then running the same test case in the normal manner.
- By counting the number of I/O operations involved in a program and computing the nominal time for these from the computer instruction manual.
- Benchmarking a program with timers and counters during IOT&E (operational environment).

Depending on the purpose for which the software reliability measurement is to be used, it may be necessary to modify the direct execution time based metric that was introduced in the preceding paragraph. Execution time can be dispensed with entirely when reliability measurements are being carried out to track the progress of a given software package during a test or modification program. Since only a measure of relative improvement is desired, and since the execution time of the program will be reasonably constant, the failure ratio rather than failure rate can be used. The failure ratio is computed by dividing the number of runs that failed by the number of successful runs during a specified time interval, e.g., one week or one month. This method can be used as a primitive form of software reliability estimation (the failure ratio rather than the failure rate is being estimated). The advantage of this variant is that it can be implemented in practically any computing environment whereas execution time based measurements require an operating system that logs execution time. The major disadvantage is that the failure ratio cannot be used for comparison among programs of different size or running on different computers because it is not self-normalizing.

3.2.2.2.2 Failures Per Execution

The failure rate based on execution time is a meaningful number that can be used for global comparisons if applied to computers of a given class, e.g., 32-bit machines in the 5 MIPS range (million instructions per second). The failure rate is not suitable for comparisons among computers of different word formats or performance classes. It is misleading to compare the failure rate on a 16-bit avionics computer that executes at 2 MIPS with that of a 60-bit mainframe executing at 20 MIPS. The latter machine processes approximately 40 times as much information in a given time interval, and if the identical test cases were run on it (only theoretically possible) the observed failure rate would have been 40 times that on the avionics computer.

For global comparisons involving computers that differ significantly in performance, it is necessary to divide the execution time based failure rate by the number of bits executed per second on each of the computers. A 16-bit computer operating at 2 MIPS executes 32 megabits per second, and the 60-bit computer operating at 20 MIPS executes 1200 megabits per second. These factors transformed the time-based failure rate into a failure rate based on information processed, i.e., failures per executions. The latter usually has little meaning in an operational environment and should be used only for research or global comparisons. Another form of this same type of measurement is failures per instructions processed.

Thus many basic units of measurement for reliability have been considered including fault density, failure rate (both execution time and computer time based), failure ratio (information processed or instructions processed). Further discussions of alternative failure rate reliability measures can be found in [THIB84].

3.2.2.3 A Proposed Structure

Our choice as a principal unit of measure for expressing software reliability is the failure rate. However, early in the development phases, the available data is more applicable to predicting a fault density. Our approach is to predict a fault density based on measurements taken early in the development phase, develop a transformation function to interpret that fault density as a predicted failure rate, and then during the later phases of development (testing) use an estimation based on failure rate. A basic measurement model is illustrated in Figure 3-3, where we recognize that software fails because it has faults (fault density represents the number of faults in the software based on its quality) and because of the environment in which it will be used (trigger rate represents the variability of inputs, the severity of the operational environment, etc). The transformation function between fault density and failure rate was developed through empirical analyses and is presented in Section 5.

3.3 RELATIONSHIP OF CANDIDATE METRICS TO STRUCTURE

With this view of software reliability, the candidate measurements (metrics) discussed earlier in this section and new measurements identified during this research effort can be organized as follows.

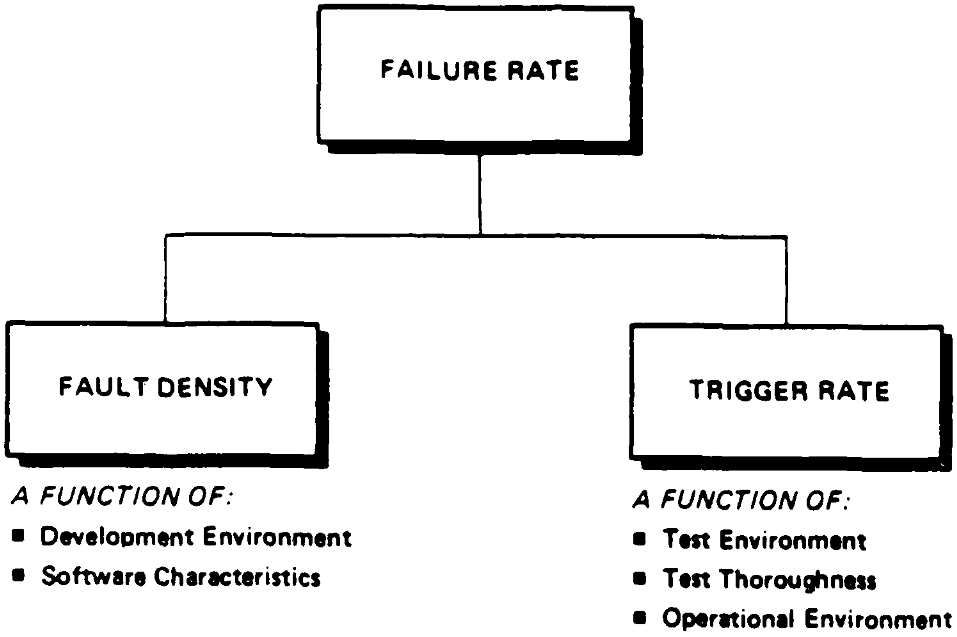


FIGURE 3-3. MEASUREMENT STRUCTURE

Those measurements which can be applied early in the development and represent an assessment of the quality of the software can be related to a measure of fault density and eventually transformed to a predictive failure rate.

Those measurements which are applied late in the development and represent an assessment of the performance of the software during testing can be related to the trigger rate.

Table 3-2 illustrates the allocation of candidate measurements to a predictive reliability number and a reliability estimation number. The measurements shown are described in the following paragraphs. Data collection procedures for each metric are in an Appendix B to Volume II of this report.

In order to maintain consistent terminology, the following conventior ; will be followed:

- The Predictive Reliability Figure-of-Merit (RP) and the Reliability Estimation Number (RE) will be called reliability numbers.
- Metrics or measures are derived values which when multiplied together will calculate one of the reliability numbers. A metric can be a simple metric (e.g., D, Development Environment) or a composite metric (e.g., S, Software Characteristics) which is the product of more than one simple metric.
- Data items are specific data elements which must be collected or measured in order to derive a metric. The data items associated with each metric are described in the Data Collection Procedures and worksheets in Appendices B and C to Volume II.

In all cases, metric values were derived from data collection and statistical analyses performed on past projects or during latter phases of this research project.

3.3.1 Predictive Metrics

In the past, software quality metrics have not met with wide acceptance because there are a large number of them, they are expensive to collect (manually), and they have not all been validated. In order to avoid these problems the following approach was adopted on this study:

- The software quality metrics (see Table 3-1) were reviewed to determine which metrics were predictive in nature. Many of the metrics currently defined in the Software Quality Measurement Framework are in effect standards, i.e., if the metric or metric worksheet item has a low score it should be corrected. These metrics are used in just that way by practioners, as QA or IV&V checklists, to

TABLE 3-2. PREDICTIVE AND ESTIMATION METRICS

PREDICTIVE METRICS			
APPLICATION TYPE	A		
DEVELOPMENT ENVIRONMENT	D		
SOFTWARE CHARACTERISTICS	S		
REQUIREMENTS AND DESIGN REPRESENTATION		S1	
ANOMALY MANAGEMENT			SA
TRACEABILITY			ST
QUALITY REVIEW RESULTS			SQ
SOFTWARE IMPLEMENTATION		S2	
LANGUAGE TYPE			SL
PROGRAM SIZE			SS
MODULARITY			SM
EXTENT OF REUSE			SU
COMPLEXITY			SX
STANDARDS REVIEW RESULTS			SR

$R_p = A \cdot D \cdot S$ WHERE

$S = S1 \cdot S2$

$S1 = SA \cdot ST \cdot SQ$

$S2 = SL \cdot SS \cdot SM \cdot SU \cdot SX \cdot SR$

ESTIMATION METRICS		
FAILURE RATE DURING TESTING	F	
TEST ENVIRONMENT	T	
TEST EFFORT		TE
TEST METHODOLOGY		TM
TEST COVERAGE		TC
OPERATING ENVIRONMENT	E	
WORKLOAD		EW
INPUT VARIABILITY		EV

$R_E = F \cdot T$, DURING TESTING WHERE

$T = TE \cdot TM \cdot TC$ and

$R_E = F \cdot E$, DURING OT&E WHERE

$E = EW \cdot EV$

report problems.

- The metrics which were considered predictive were retained.

- The metrics which were considered to be QA/IV&V checklists candidates are advocated as review checklists to be used during formal reviews such as design reviews and informal reviews such as walkthroughs.

- The number of problem reports generated as a result of applying these checklists is a metric to be used.

Several new metrics were identified also and are discussed in the following paragraphs. The Predictive Reliability Figure-of-Merit (Rp) is the product of the identified metrics. The individual metrics were adjusted during validation to a numeric that can be used as a multiplier in this product. The final results are presented in Volume II. The validation process is described in Section 5 of this Volume.

3.3.1.1 Application Type (A)

The type of application, i.e., the function to be performed, is considered a basic characteristic of the software. It is considered in this study as the basis for establishing a nominal prediction number. The type of application typically affects both the manner in which software is developed and how it is operated. Because of those affects, the application type is not independent of the other metrics to be discussed. However, since it is perhaps the first characteristic known about the software it is a valuable initial predictor. Our concept is to use a classification scheme for the application type. A fault density (or failure rate) will be associated with each category or application type. We will develop that metric by looking at a wide range of systems and taking the average for those that fall within each application type. The metric will be a fault density associated with the application type chosen, A.

Several potential classification schemes were identified. They are presented in Table 3-3. For the sake of this study, we decided to evaluate two of these approaches. Hecht's basic categorization was real-time, interactive, batch processing and support. He further distinguishes each of these categories depending on access. In [MCCA77], an application scheme that was Air Force application-related was developed. This scheme was developed to be oriented toward the AF SAM or SPO. The RCA PRICE-S model uses the classification scheme in column three for the parameter PLATFORM recognizing the influence of Military Standards on a system. The PRICE-S model also uses an application mix for the software. The categorization scheme for this mix plus the relative numerics used in the PRICE-S system are shown in Table 3-4. The RADC Test Handbook [PRES84] uses the

TABLE 3-3. CANDIDATE APPLICATION CLASSIFICATION SCHEMES

HECHT (HECH84)	McCALL (MCCA77)	RCA (PRIC77)	BOEING (PREI84)
• REAL TIME OPERATING SYSTEM • REAL TIME CLOSED LOOP OPERATING SYSTEM • OTHER REAL TIME • INTERACTIVE OPERATING SYSTEM • INTERACTIVE APPLICATION – PUBLIC • INTERACTIVE APPLICATION – RESTRICTED • SCIENTIFIC BATCH • OTHER BATCH • SUPPORT PROGRAM • HARDWARE DIAGNOSTIC • SOFTWARE TOOLS AND DIAGNOSTICS • OTHER	• MANNED SPACECRAFT AIRBORNE AVIONICS • UNMANNED SPACECRAFT MISSILES • INDICATION AND WARNING • SENSOR DATA PROCESSING/ INTELLIGENCE • STRATEGIC/ TACTICAL C^2 • COMMUNICATIONS • MIS • DEVELOPMENT/ TEST BED	• MANNED SPACECRAFT • UNMANNED SPACECRAFT • MIL SPEC AVIONICS • COMMERCIAL AVIONICS • MOBILE SYSTEM • NON REAL TIME C^2 • MIL SPEC GROUND SYSTEM • SATELLITE GROUND SYSTEM • PRODUCTION CENTER SOFTWARE – CONTRACTOR DEVELOPED • PRODUCTION CENTER SOFTWARE – USER DEVELOPED	• BATCH • EVENT CONTROL • PROCESS CONTROL • PROCEDURE CONTROL • NAVIGATION • FLIGHT DYNAMICS • ORBITAL DYNAMICS • MESSAGE PROCESSING • DIAGNOSTIC SOFTWARE • SENSOR & SIGNAL PROCESSING • SIMULATION • DBMS • DATA ACQUISITION • DATA PRESENTATION • DECISION & PLANNING AIDS • PATTERN & IMAGE PROCESSING • COMPUTER SYSTEM SOFTWARE • SOFTWARE DEVELOPMENT TOOLS

TABLE 3-4. SYSTEM TYPE CATEGORIZATION

SYSTEM TYPE	RELATIVE NUMERIC
PRODUCTION CENTER SOFTWARE - DEVELOPED BY USER	0.8
PRODUCTION CENTER SOFTWARE - DEVELOPED BY CONTRACTOR	1.0
SATELLITE GROUND SYSTEM	1.0
MIL-SPEC GROUND SYSTEM	1.2
NON-REAL-TIME COMMAND AND CONTROL	1.2
MOBILE SYSTEM (VAN SHIPBOARD)	1.4
COMMERCIAL AVIONICS	1.7
MIL-SPEC AVIONICS	1.8
UNMANNED SPACECRAFT	2.0
MANNED SPACECRAFT	2.5
APPLICATION MIX	**RELATIVE NUMERIC**
DATA STORAGE AND RETRIEVAL	4.5
ON-LINE COMMUNICATIONS	6.8
REAL-TIME COMMAND AND CONTROL	9.4
INTERACTIVE OPERATIONS	12.1
MATHEMATICAL APPLICATIONS	1.0
STRING MANIPULATION	2.5
OPERATING SYSTEMS	12.1

classification scheme in column four. This categorization relates specifically to the functions being performed by the software. From a system perspective, there are typically a number of these functions being performed within a system. The two approaches chosen for evaluation were the first two. Each was modified as shown in Table 3-5.

The Air Force application scheme has six major categories: airborne, strategic, tactical, process control, production center, and developmental/support. Airborne applications are systems which perform real-time closed loop functions such as navigation, flight control, fire control, and electronic warfare on-board an aircraft. Systems on-board a satellite performing orbital control, data acquisition, and power supply control would also be considered airborne systems. Strategic applications are systems involved in planning, directing or providing warning of large-scale military operations. An industry equivalent application would be a company wide communication system supporting business management, decision support, and operation. Indication and warning systems like a ballistic missile defense system are considered a strategic application. Tactical applications are systems involved in support of actual enemy engagements providing such functions as weapon system fire control, short range communications, and combat decision support. Process Control applications are systems involved in monitoring and controling machinery such as numerical control manufacturing equipment and nuclear power plants. The production center application category involves Managment Information Systems such as personnell, finance, payroll, inventory control that typically run in a computer center environment primarily in batch mode. More modern examples of these types of systems are on-line interactive transaction processing systems. The Developmental Support applications category includes those systems which support the development of systems (eg. software engineering environments), simulations, testbeds, and analytical packages. Examples of systems which would fall in such categories is shown in Table 3-5. These examples serve as definitions of the categories. The time dependence scheme has four basic categories of real-time, on-line interactive or transaction processing, batch, and support software. We considered subcategorizing real-time into close-loop (eg. flight control) and other and on-line into distributed and centralized to evaluate the differences of those subcategories but postponed that for future research.

Table 3-5A identifies a categorization scheme based on software function [PRES84] that is recommended for future research. This more detailed categorization scheme would provide a nominal (baseline) reliability at a subsystem or CPC level.

Where more detailed information is available, we could further categorize the application by that set of software functions being performed and the time dependency of these functions. We anticipate that we will eventually, based on observed data,

TABLE 3-5 APPLICATION CLASSIFICATION SCHEMES

APPLICATION	TIME DEPENDENCE
• AIRBORNE SYSTEMS - MANNED SPACECRAFT - UNMANNED SPACECRAFT - MIL-SPEC AVIONICS - COMMERCIAL AVIONICS • STRATEGIC SYSTEMS - C^3I - STRATEGIC C^2 - INDICATIONS AND WARNING - COMMUNICATIONS • TACTICAL SYSTEMS - TACTICAL C^2 - TACTICAL MIS - MOBILE - EW/ECCM • PROCESS CONTROL SYSTEMS - INDUSTRIAL PROCESS CONTROL • PRODUCTION SYSTEMS - MIS - DECISION AIDS - INVENTORY CONTROL - SCIENTIFIC • DEVELOPMENTAL SYSTEMS - SOFTWARE DEVELOPMENT TOOLS - SIMULATION - TEST BEDS - TRAINING	• REAL-TIME • ON-LINE (INTERACTIVE/TRANSACTION PROCESSING) • NON-TIME CRITICAL (BATCH) • SUPPORT

TABLE 3-5A. APPLICATION CLASSIFICATION SCHEMES

FUNCTION
• EVENT CONTROL
• PROCESS CONTROL
• MESSAGE PROCESSING
• SENSOR AND SIGNAL PROCESSING
• PATTERN AND IMAGE PROCESSING
• DISTRIBUTION/COMMUNICATION
• DISPLAY/DATA PRESENTATION
• PROCEDURE CONTROL
• RESOURCE MANAGEMENT/CONTROL
• SCIENTIFIC/ANALYTICAL PROCESSING
• DECISION AND PLANNING AIDS
• DATA MANAGEMENT
• EXECUTIVE/OPERATING SYSTEM
• SUPPORT SOFTWARE/UTILITIES
• DIAGNOSTICS

modify several of the categories.

As the development proceeds the nominal predicted reliability for the application will be modified based on the development environment, the characteristics exhibited by the software as it evolves, and its performance during testing. This is analogous to the procedure used for hardware reliability prediction where initially a nominal parts failure rate is assigned which is modified by quality, derating, and environment factors as the design is definitized.

3.3.1.2 Development Environment (D)

This metric is concerned with effects of the development environment on the reliability of the software produced within that environment. In the development of the COCOMO software cost model, Boehm found that there were significant differences between three classes of environments which he termed organic, semi-detached, and embedded [BOEH81]. It is expected that these environment characteristics will also affect software reliability.

The following descriptions of each of the environments and the table of distinguishing features (Table 3-6) are excepted from the cited reference.

> ORGANIC MODE - In the organic mode, relatively small software teams develop software in highly familiar, in-house environments. Most people connected with the project have extensive experience in working with related systems within the organization, and have a thorough understanding of how the system under development will contribute to the organization's objectives.
>
> SEMIDETACHED MODE - The semidetached mode of software development represents an intermediate stage between the organic and embedded modes. The team members all have an intermediate level of experience with related systems. The team has a wide mixture of experienced and inexperienced people, and team members have experience related to some aspects of the system under development, but not to others.
>
> EMBEDDED MODE - The major distinguishing factor of an embedded mode software project is a need to operate within tight constraints. The product must operate (is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures such as electronic funds transfer system or air traffic control system. In general the costs of changing the other parts of this complex are so high that their characteristics are considered essentially unchangeable, and the software is expected both to conform to their specifications and to take up the slack of any unfore-

TABLE 3-6. DISTINGUISHING FEATURES OF SOFTWARE DEVELOPMENT MODES (BOEH81)

	MODE		
FEATURE	ORGANIC	SEMIDETACHED	EMBEDDED
ORGANIZATIONAL UNDERSTANDING OF PRODUCT OBJECTIVES	THOROUGH	CONSIDERABLE	GENERAL
EXPERIENCE IN WORKING WITH RELATED SOFTWARE SYSTEMS	EXTENSIVE	CONSIDERABLE	MODERATE
NEED FOR SOFTWARE CONFORMANCE WITH PRE-ESTABLISHED REQUIREMENTS	BASIC	CONSIDERABLE	FULL
NEED FOR SOFTWARE CONFORMANCE WITH EXTERNAL INTERFACE SPECIFICATIONS	BASIC	CONSIDERABLE	FULL
CONCURRENT DEVELOPMENT OF ASSOCIATED NEW HARDWARE AND OPERATIONAL PROCEDURES	SOME	MODERATE	EXTENSIVE
NEED TO INNOVATE DATA PROCESSING ARCHITECTURES, ALGORITHMS	MINIMAL	SOME	CONSIDERABLE
PREMIUM ON EARLY COMPLETION	LOW	MEDIUM	HIGH
PRODUCT SIZE RANGE	< 50 KDSI	< 300 KDSI	ALL SIZES
EXAMPLES	BATCH DATA REDUCTION SCIENTIFIC MODELS BUSINESS MODELS FAMILIAR OS, COMPILER SIMPLE INVENTORY, PRODUCTION CONTROL	MOST TRANSITION PROCESSING SYSTEMS NEW OS, DBMS AMBITIOUS INVENTORY, SIMPLE COMMAND CONTROL	LARGE, COMPLEX TRANSITION PROCESSING SYSTEMS AMBITIOUS VERY LARGE OS AVIONICS AMBITIOUS COMMAND-CONTROL

seen difficulties.

A metric, D_1, will be associated with each of these three environments. That metric will be modified based on further distinguishing characteristics shown in Table 3-7. These characteristics further distinguish the level of formality, discipline, and modern approach to the development effort [SOIS85]. The characteristics will be in the form of a checklist which will be used to score the development enviroment. The score will modify the initial environment metric, D_1, resulting in the metric D. This resulting metric, D, will be a multiplier of the fault density associated with the Application Type and affect it positively (the multiplier will be less than one but greater than zero) or negatively (the multiplier will be greater than one), thus representing the positive or negative effect the development environment has on the production of reliable software.

3.3.1.3 Software Characteristics (S)

This set of metrics represent those characteristics of the software which are likely to affect the software reliability. The characteristics can be measured from the code and the documentation produced during the software development process. The metrics within this set are further organized, for recognition purposes, under Requirements and Design Representation metrics and Software Implementation metrics. Those metrics in the former group are applied to the documentation which represents the software requirements of the system and the software design. They will typically be applied at the time of formal reviews such as the Software Requirements Review (SRR), the Preliminary Design Review (PDR) and the Critical Design Review (CDR). Those metrics in the latter group are applied to the code during the coding phase of the development. Each metric is described in the following paragraphs.

3.3.1.3.1 Requirements and Design Representation Metrics (S1)

- Anomaly Management (SA)

  This metric represents the degree to which fault tolerance has been designed and implemented in the system. The ability of the software to accept anomalous input data, recover from incorrect calculations, gracefully degrade, and fail in a controlled manner contributes to its reliability. Various strategies for developing error tolerance software exist [MYER76]. A checklist approach to evaluating these features was first proposed by [MCCA77] and expanded by [BOWE83]. The features assessed include:

  - Error Condition Control

  - Input Data Checking

TABLE 3-7. DISTINGUISHING CHARACTERISTICS OF DEVELOPMENT ENVIRONMENT (Modified from [SO1585])

ORGANIZATIONAL/PERSONNEL CONSIDERATIONS

Separate Design and Coding
Independent Test Organization
Independent Quality Assurance
Independent Configuration Mangement
Independent Verification and Validation
Chief Programming Teams
Above Average Educational Level of Team Members
Above Averrage Experience Level of Team Members

METHODS USED

Definition/Enforcement of Standards
Use of HOL
Formal Reviews (SRR, PDR, CDR, etc.)
Frequent Walkthroughs
Top Down and Structured Approaches
Unit Development Folders
Software Development Library
Formal Change and Error Reporting
Progress and Status Reporting

DOCUMENTATION

System Requirements Specification
Software Requirements Specification
Interface Design Specification
Software Design Specification
Test Plans, Procedures and Reports
Software Development Plan
Software Quality Asssurance Plan
Software Configuration Management Plan
Requiremetns Traceability Matrix
Version Description Document
Software Discrepancy Reports

DEVELOPMENT TOOLS

Requirements Specification Language
Program Design Language
Program Design Graphical Technique (Flowchart, HIPO, etc)
Simulation /Emulation
Configuration Management
Code Auditor
Data Flow Anallyzer
Quality Measurement Tools

- Computational Failure Identification and Recovery
- Hardware Fault Identification and Recovery
- Device Error Identification and Recovery
- Communication Failure Identification and Recovery

The metric, SA, is:

$$SA - ka/AM$$

where ka is a coefficient to be derived from regression and AM is the evaluated score from application of the checklists in [BOWE83] (metrics AM.1, AM.2, AM.3, AM.4, AM.5, AM.6, AM.7, RE.1).

The checklists have been modified somewhat during the process of use/experience during this effort. They are presented in the Data Collection Procedures, Appendix B of Volume II of this report.

- Traceability (ST)

The traceability metric is based on an identically named criterion in [MCCA80] and [BOWE85]. The metric used there, the cross reference relating modules to requirements, will also be applied to the current study. The basic concept of this criterion is that if the requirements are traceable to the code then there is less of a chance that a misinterpretation of the requirements can result in a fault in the code.

The effect on reliability will be represented by the traceability metric, ST, as:

$$ST - k_{tc}/TC$$

where k_{tc} represents a coefficient to be determined by regression and TC is the traceability metric (TC.1) in Table 3-1, which is calculated by identifying the total number of requirements (NR) and dividing this number by the total number of traceable requirements (NR-DR) where DR is the number of requirements not traceable to design or code. A methodology for itemizing requirements can be found in [HERN83] or use of tools/techniques such as SREM [BELL76] or PSL/PSA [TEIC76] also support this type of calculation. A further description of how to calculate the metric is in Volume II of this report.

- Quality Review Results (SQ)

During most large system developments various formal reviews are conducted. Previously mentioned examples such as SRR, PDR, CDR are typical formal reviews. Informal reviews, audits, or inspections may also be conducted. Two such techniques are structured walkthroughs and design and code inspections [FAGA76]. The quality of the documentation and the design represented by the documentation is reviewed during these activities. Any problems identified are recorded as a problem report or action item for correction. Studies have shown that the more problems encountered early in a development the more likely it is that problems will exist and be found later during test and operation [LIPO79]. This metric, Quality Review Results (SQ), represents a measure of the number of problem reports or discrepancies reported during reviews. The metric takes the following form:

$$SQ = k_q * (NR/NR-NDR)$$

where k_q is a coefficient derived from regression (see Section 5), NDR is the number of discrepancy reports identified, and NR is the total number of requirements identified in the system.

Use of the worksheets (checklists) in Appendix D of Volume II is advocated. These worksheets contain data elements related to the software quality metrics in Table 3-1:

Accuracy	(AC.1)
Completeness	(CP.1)
Consistency	(CS.1, CS.2)
Autonomy	(AU.1, AU.2)

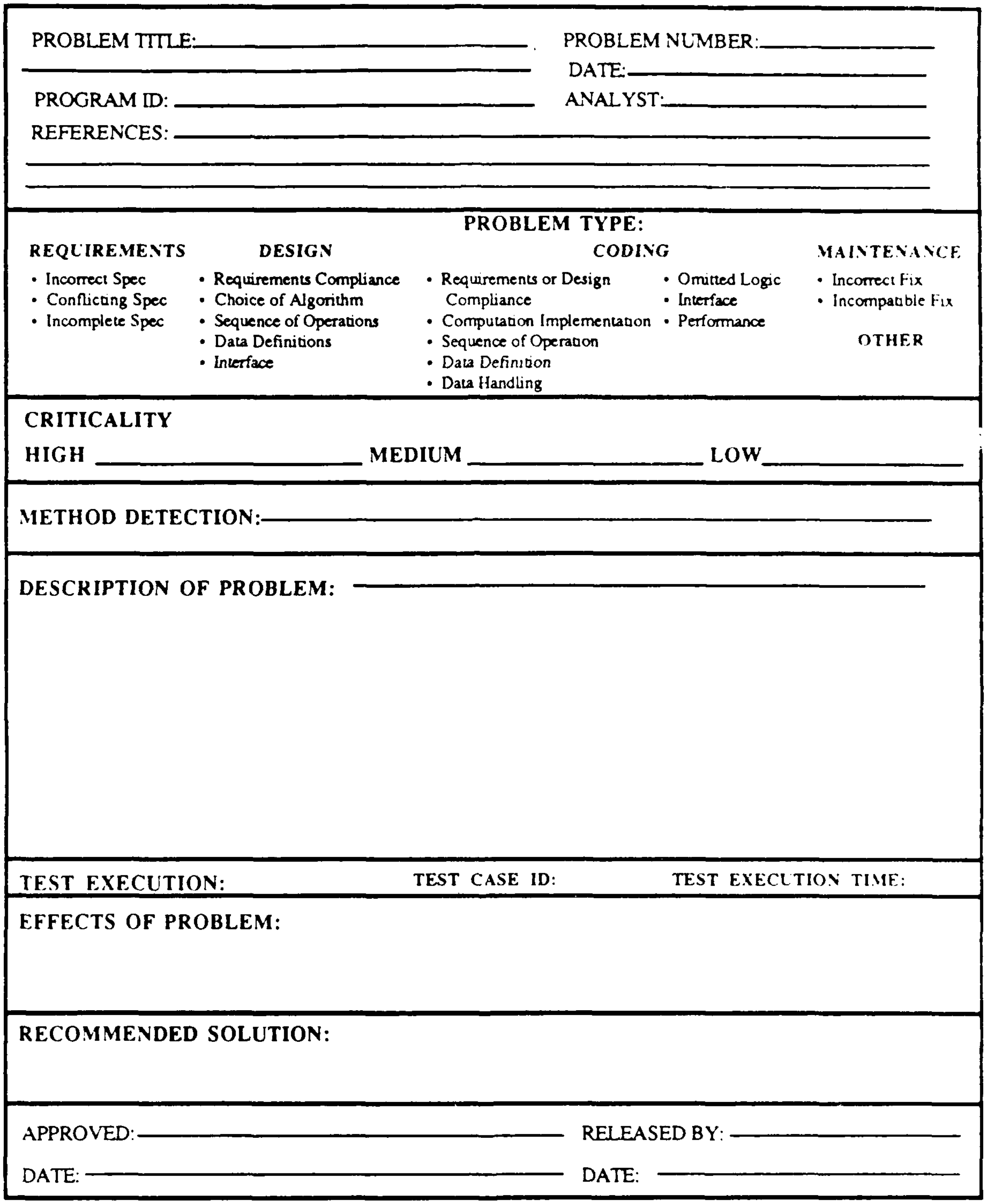
A discrepancy report should be generated for each question on these worksheets answered negatively when applicable. An example discrepancy report is shown in Figure 3-4.

The worksheets assess how well the following characteristics have been addressed in the requirements and design of the system.

- Accuracy - the concept of reliability includes precision, i.e., algorithms must be accurate within certain bounds.

- Completeness - the requirements and design should have the following characteristics:

  -- Unambiguous references,



PROBLEM TITLE: ______________________ PROBLEM NUMBER: ____________

______________________ DATE: ____________

PROGRAM ID: ______________________ ANALYST: ____________

REFERENCES: ______________________

____

____

PROBLEM TYPE:

REQUIREMENTS

- Incorrect Spec
- Conflicting Spec
- Incomplete Spec

DESIGN

- Requirements Compliance
- Choice of Algorithm
- Sequence of Operations
- Data Definitions
- Interface

CODING

- Requirements or Design Compliance
- Computation Implementation
- Sequence of Operation
- Data Definition
- Data Handling
- Omitted Logic
- Interface
- Performance

MAINTENANCE

- Incorrect Fix
- Incompatible Fix

OTHER

CRITICALITY

HIGH ____________ **MEDIUM** ____________ **LOW** ____________

METHOD DETECTION: ______________________

DESCRIPTION OF PROBLEM: ______________________

TEST EXECUTION: **TEST CASE ID:** **TEST EXECUTION TIME:**

EFFECTS OF PROBLEM:

RECOMMENDED SOLUTION:

APPROVED: ______________________ RELEASED BY: ______________________

DATE: ______________________ DATE: ______________________

FIGURE 3-4 DISCREPENCY REPORT

-- All data references defined, computed, or obtained from an external source,

-- All defined functions used,

-- All referenced functions defined,

-- All conditions and processing defined for each decision point,

-- All defined and referenced calling parameters agree, and

-- All discrepancy reports resolved.

- Consistency - the requirements and design should have:

-- Standard design representation,

-- Calling sequence conventions,

-- Input/output conventions,

-- Data naming conventions, and

-- Error handling conventions.

- Autonomy - the software components should be independent functions and as non-dependent of their interfaces as possible.

In order for this metric to take on true significance, statistical studies of projects employing similar review concepts or at least devoted similar levels of effort to reviewing the requirements and design will have to be conducted. Projects employing IV&V contractors would be applicable subjects.

3.3.1.3.2 Software Implementation Metrics (S2)

- Language Type (SL)

The programming language chosen and used to implement a system can have an effect on the reliability of the system. A significant dependency of fault density on language has been established in [HECH83].

The metric for Language (SL) will be based on the classification, identified as:

- Assembly level programs, and

- Higher-order language programs.

The HOL category will represent the default (to be assigned a value of 1). It has been assumed that one HOL statement will generate machine instructions equivalent of two to eight assembly statements. Five is a typical expansion ratio for FORTRAN. Under these circumstances the metric is:

$$SL(\text{Assembly}) = 1.4$$

$$SL(\text{HOL}) = 1$$

Where programs contain a mixture of HOL and assembly language code, the language criterion is computed as the sum of the fractions applicable to each category. Thus, for a mixed language program, the language metric, SL, is given by

$$SL = (\text{HOL\%}) * 1 + (\text{Assembly \%}) * 1.4$$

- Program Size (SS)

This metric represents the effect of total size on reliability. We already stated that the failure rate measure of reliability is self-normalizing with respect to size, however we feel there are secondary effects which should be taken into account. These secondary effects are associated with inherent complexity, number of interactions, data base size and the ability of humans to deal with extremely large systems.

The metric will be a multiplier associated with size categories (or ranges). Tentatively size categorizations to be used are:

$$\begin{aligned} & SS(1) < 10000 \text{ lines of code} \\ 10000 < \; & SS(2) < 50000 \text{ lines of code} \\ 50000 < \; & SS(3) < 100000 \text{ lines of code} \\ 100000 < \; & SS(4) \end{aligned}$$

In this case, lines of code are defined as all executable source statements.

- Modularity (SM)

It is generally held that small modules can be more readily reviewed and are, therefore, less likely to contain faults than larger modules (this is implicit in MIL-STD-1679). It is intended to establish three categories for module size, based on the number of executable statements:

$$\begin{aligned} & SM(1) < 200 \text{ lines of code} \\ 200 < \; & SM(2) < 3000 \text{ lines of code} \end{aligned}$$

3000 < SM(3)

For the assessment of software development practices it might be of interest to apply this metric to individual modules and to correlate it with failures due to these modules. In many cases, available data from historical projects do not support an analysis at this detailed level. Regardless of data quality, it is frequently impossible to associate a specific module with a software failure (e.g., for failures due to missing requirements, faulty interface specifications or implementations). For cases where detailed data is available, the metric will be evaluated by the following:

$$SM = (u*SM(1) + v*SM(2) + w*SM(3)) / (u+v+w)$$

where SM is the overall module size metric, lower case letters are the number of modules in a given category and upper case letters are the module size coefficients applicable to each category.

For the purpose of reliability prediction, for this study, it is considered adequate to base the metric for module size on the average size in a program (i.e., total executable statements divided by the number of modules). The metric, SM, applicable to each module size classification was evaluated by regression (see Section 5).

- Extent of Reuse (SU)

As the application of computers to Air Force projects matures, there are increasing opportunities for including portions of operational code in new software developments. The practice appears desirable for reliability as well as for economic reasons Code from current operational programs is expected to contain fewer faults than newly generated code since through previous test and maintenance efforts its reliability will have grown to an acceptable level. The reliability of the current code is assumed to be known by observation during operation.

However, it is important to recognize any differences in environment, application, or interfaces that the existing software may encounter will have a potential impact on its reliability. In the situation where new code is being added to existing code in the same environment, the existing code's reliability can be taken as observed. In the situation where the existing code is being used in a new environment as part of the development of a new application, it cannot be expected, without analysis, to perform with its established reliability because of new requirements and interfaces. In each case, though, the failure rate for the reused code should be less than that for the new code. The metric for reused code (SU) in reliability prediction will be:

$$SU = SU(i)$$

where Su (i) is a factor derived from empirical data.

Initially we expect this factor to be determined by looking up a factor in a Table with data from a limited number of projects.

- Complexity (SX)

Candidate metrics include the SI.3 and SI.4 metrics from [BOWE85] (see Table 3-1). SI.3 is McCabe's cyclomatic complexity metric [MCCA76] and SI.4 is the checklist assessing the simplicity with which a program is implemented. Halstead's metrics (SI.6) should also be considered [HALS77]. Past experience applying these metrics indicates McCabe's metric to be more applicable because it can be automatically calculated and has demonstrated better correlation than Halstead's metric. [MCCA80].

Since this metric is applied when the project is close to entering the reliability estimation phase, prediction that accounts for complexity may be helpful in several ways:

- It will identify the role that complexity plays in causing failures (by use of regression techniques).

- It will encourage recording of complexity measures as part of the project history.

- By virtue of the above it will identify long range trends of increasing or decreasing complexity which may not otherwise be captured in an analysis of software failures.

This metric is applicable at the module level. Again, the availability of data at this level may hinder the establishment of a prediction coefficient and use of the metric during projects. When available the metric (SX) will be:

$$SX = k_x * \left(\sum_{i=1}^{n} SX_i \right) / n$$

where SX_i is McCabe's complexity (SI.3 in Table 3-1) for each module, i, in the system, n equals the total number of modules in the system, and k_x is a coefficient derived from regression.

- Standards Review Results (SR)

As during requirements and design, reviews, audits, inspections and walkthroughs are techniques for identifying discrepancies or problems to be corrected. This metric represents the number of problems identified per module based on reviews or audits of the code.

Worksheets from software quality metrics (SI.1, SI.2, SI.4, SI.5, MO.1, MO.2) are advocated. Enforcement of programming standards is another technique when discrepancies would be identified. Worksheets are in Appendix D of Volume II. The overall metric then will be a composite, based on the evaluation of the following characteristics:

- Design organized in top-down fashion,
- Independence of module,
- Module processing not dependent on prior processing,
- Each module description includes input, output, processing, limitations,
- Each module has a single entrance, single exit,
- Size of data base,
- Compartmentalization of data base,
- No duplicate functions, and
- No global data.

The metric will be:

$$SR = kv * (n/n-PR)$$

where n = number of modules
PR = number of problem modules identified with severe discrepancies
kv = coefficient derived by regression

Classification of the types of problems being identified can be helpful. Three problem classification schemes are shown in Table 3-8. The middle column, has been used most widely in the past. The right hand column is the one advocated primarily because of its development phase orientation. By looking at the types of errors being identified, standards can be improved, checklists can be improved, and development techniques can be improved to help avoid making similar errors in the future.

TABLE 3-8 ERROR CLASSIFICATIONS

GOEL [GOEL83]	TRW [THAY76]	JLC [JLC81]
• SYNTAX • SEMANTIC • RUNTIME • DOMAIN • COMPUTATIONAL • NON-TERMINATION • SPECIFICATION • PERFORMANCE	• COMPUTATIONAL • LOGIC • DATA DEFINITION • DATA HANDLING • DESIGN • INTERFACE • COMPOOL • PROBLEM REPORT REJECTION • OTHER • TEST-ONLY CODE • OPTIMIZATION - TIMING - SIZING • INTEGRATION OF NEW SOFTWARE • UNNECESSARY CODE • NEW REQUIREMENTS • STANDARDS VIOLATION	• REQUIREMENTS - INCORRECT SPEC - CONFLICTING SPEC - INCOMPLETE SPEC • DESIGN - REQUIREMENTS COMPLIANCE - CHOICE OF ALGORITHMS - SEQUENCE OF OPNS - DATA DEFINITION - INTERFACE • CODING - REQ OR DES COMPLIANCE - COMPUTATIONAL IMP - SEQUENCE OF OPN - DATA DEFINITION - DATA HANDLING - OMITTED LOGIC - INTERFACE - PERFORMANCE • MAINTENANCE - INCORRECT FIX - INCOMPATIBLE FIX • OTHER

3.3.1.4 Other Metrics

Two other quality metrics identified in Table 3-1, Self-Descriptiveness and Distributedness, were not used. Self-Descriptiveness seemed particularly applicable to maintainability and not appropriate for reliability prediction. Distributedness is appropriate for distributed systems and, therefore, a special case not applicable to our generic methodology.

Visibility, a quality metric identified in Table 3-1, is appropriate as an estimation metric and discussed in subsequent paragraphs.

3.3.2 Estimation Metrics

As previously discussed, the use of reliability model technology has not been widely accepted. The basic approach of this technology, observing the failure rate of the software during test, will be used within our methodology. Our approach to estimation is to observe testing and calculate the observed failure rate of the software. This basic estimation number will be adjusted based on one of two environmental metrics, T during the development test phases and E during the Operational Test and Evaluation phase. The estimation number will be the product of the observed failure rate and one of those metrics. These metrics are described in the following paragraphs.

3.3.2.1 Failure Rate During Test (F)

The basic metric for estimation will be the observed failure rate during testing (F). Reliability models have been researched for a number of years and provide a mechanism for estimation. The basic philosophy of the reliability models is illustrated in Figure 3-5 (using the Musa Model as an example) [MUSA75]. The observed number of failures over time (and therefore the mean time between failures) is extrapolated via a curve fitting exercise (using the basic assumed model) and knowing the amount of test time expended to date, one can estimate the amount of additional test time required to achieve an acceptable (estimated) failure rate. A large number of models exist. Twenty three models described in [GOEL83] are listed in Table 3-9. Experience using these models has varied ([MUSA79], [RICH83], [ANGU83]) and because of that variability, make the models suspect as estimation techniques. In lieu of their use, tracking the observed failure rate during testing provides a basis for estimation. This is illustrated in Figures 3-6 and 3-7. Figure 3-6 demonstrates the use of execution-time measures during the pre-operational (test) phase [HECH77]. The data came from the development of the Metric Integrated Processing System (MIPS) at Vandenberg Air Force Base during which disciplined programming techniques were introduced under an RADC sponsored effort. The linear regression line exhibits an improvement in reliability (reliability growth) over time (the downward slope). It also shows several significant increases in failure rate during

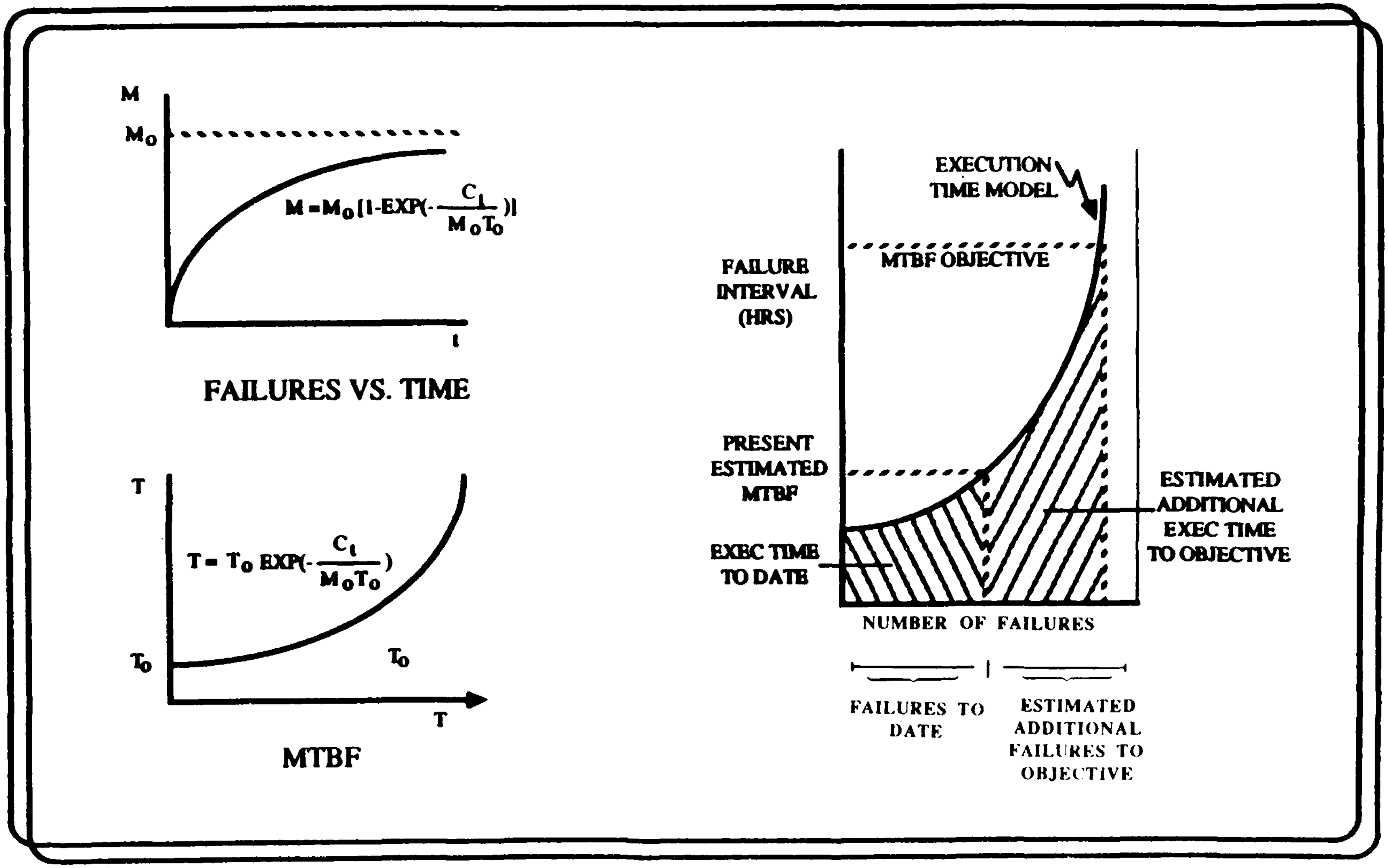


FIGURE 3-5 EXAMPLE RELIABILITY MODEL [MUSA75]

TABLE 3-9. SOFTWARE RELIABILITY MODELS [GOEL83]

TIMES BETWEEN FAILURES MODELS

- JELINSKI AND MORANDA DE-EUTROPHICATION
- SCHICK AND WOLVERTON LINEAR
- SCHICK AND WOLVERTON PARABOLIC
- GEOMETRIC DE-EUTROPHICATION
- HYBRID GEOMETRIC POISSON
- GOEL AND OKUMOTO IMPERFECT DEBUGGING
- LITTLEWOOD - VERRAL BAYESIAN

FAULT SEEDING MODELS

- MILLS SEEDING

INPUT DOMAIN MODELS

- NELSON
- HO
- RAMAMOORTHY AND BASTANI

FAILURE COUNT MODELS

- GOEL - OKUMOTO NON-HOMOGENEOUS POISSON PROCESS
- SCHNEIDEWIND
- GOEL MODIFIED NON-HOMOGENEOUS POISSON PROCESS
- MUSA EXECUTION TIME
- SHOOMAN EXPONENTIAL
- GEOMETRIC POISSON
- MODIFIED JELINSKI - MORANDA
- MODIFIED GEOMETRIC DE-EUROPHICATION
- MODIFIED SCHICK AND WOLVERTON
- GENERALIZED POISSON
- IBM BINOMIAL
- IBM POISSON

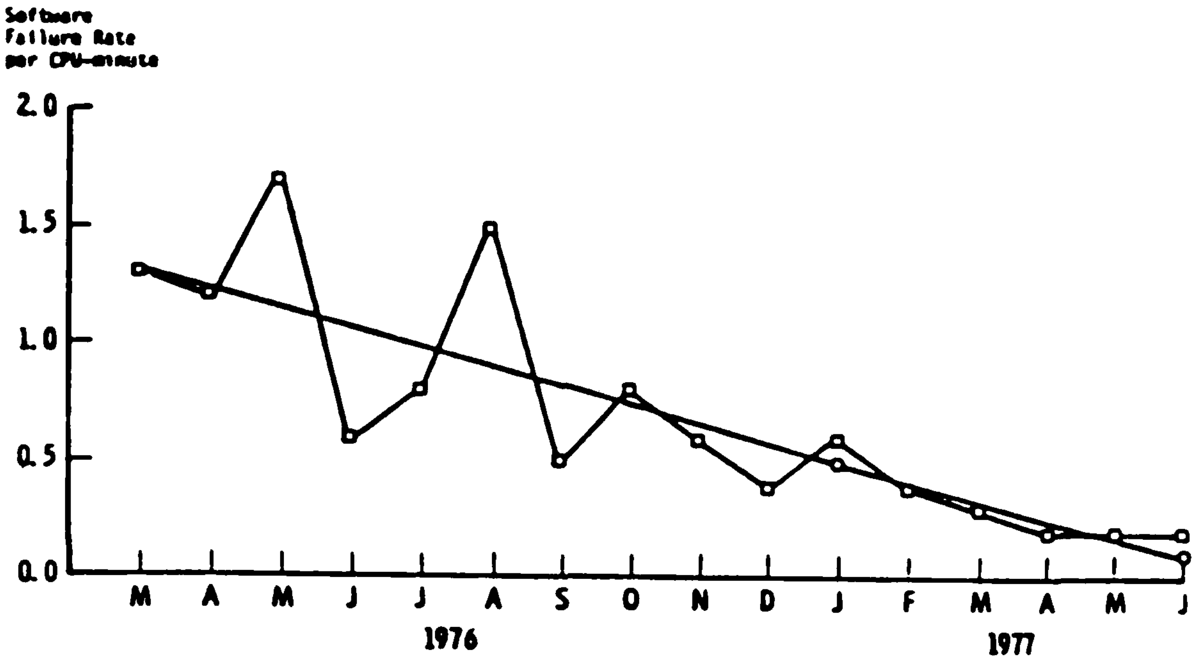


FIGURE 3-6 FAILURE RATE DURING DEVELOPMENT {VAND83}

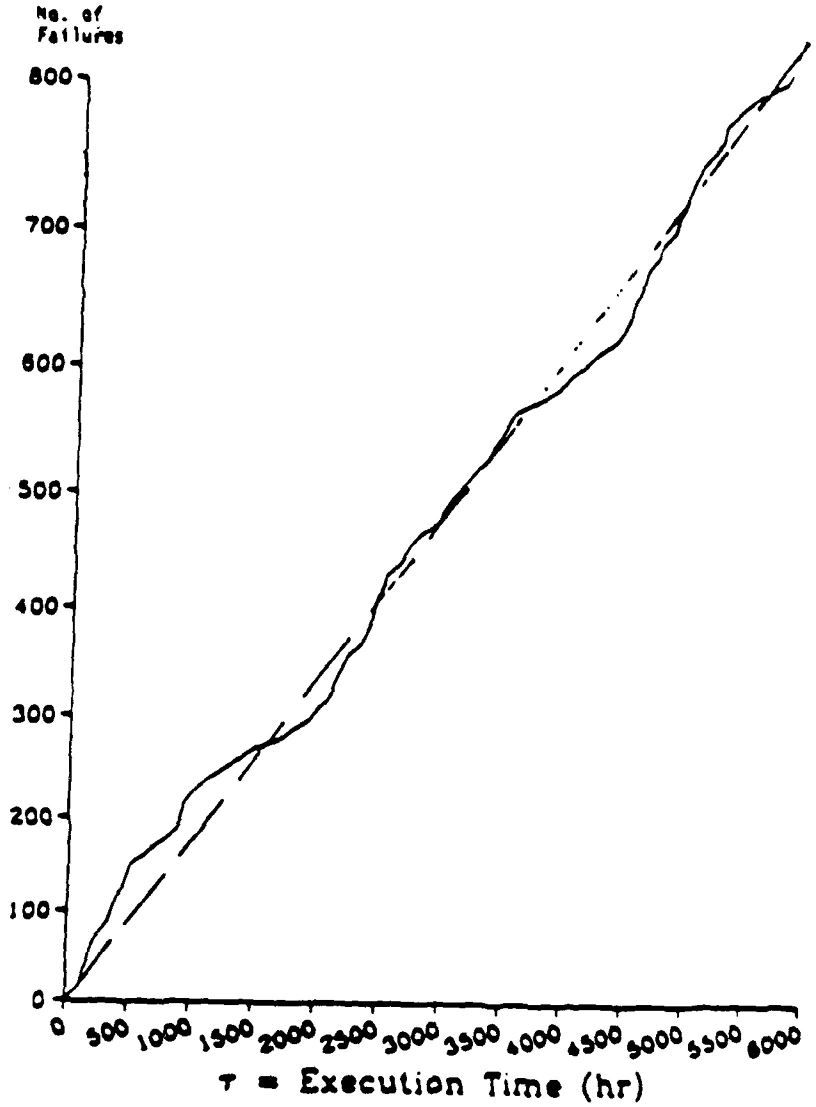


FIGURE 3-7 FAILURE EXPERIENCE DURING OPERATION {MUSA79}

specific months. In each case there was always a specific reason: In May and August 1976 major new modules were added to the system under test; in October 1976, the contractor's quality assurance organization took over responsibility for the test; and January 1977 marked the start of testing by the Air Force.

Similar consistency in time for this type of metric during operation is shown in Figure 3-7 [MUSA79]. The failure rate is indicated by the slope of the data line. Note that the ordinate scale is nonlinear in order to permit the number of failures predicted by the MUSA model to be plotted as a straight line. A last example is provided in Figure 3-8 from [ANGU79]. In this example, a consistent reliability growth was not observed. A high failure rate was still being observed at the end of the illustrated test phase.

By tracking this metric during testing, the trend in the observed failure rate can be monitored and used as the basis for estimating what the expected operational reliability will be.

3.3.2.2 Test Environment (T)

Several characteristics of the test environment should be accounted for in the estimation of reliability. The observed failure rate may not accurately represent what the operational reliability will be because:

- The test environment does not accurately represent the operational environment,
- The test data does not thoroughly exercise the system thereby leaving untested many segments of the code,
- The testing techniques employed do not thoroughly test the system, and
- The amount of testing time does not thoroughly test the system.

These characteristics are taken into account by the metrics to be discussed in this paragraph. In each case the metrics will be in the form of a multiplier, the product of all of these to be used to adjust the observed failure rate (F) up or down depending on the level of confidence in the representativeness and thoroughness of the test environment (T = TE*TM*TC).

- Test Effort (TE)

  This metric is intended to represent the amount of effort applied to testing. Three alternatives are to be evaluated. The first alternative is the test budget (dollars or labor hours) which would appear to be a good metric for the amount of test. Comparison with a guideline of 40% of total development effort would be the metric. However,

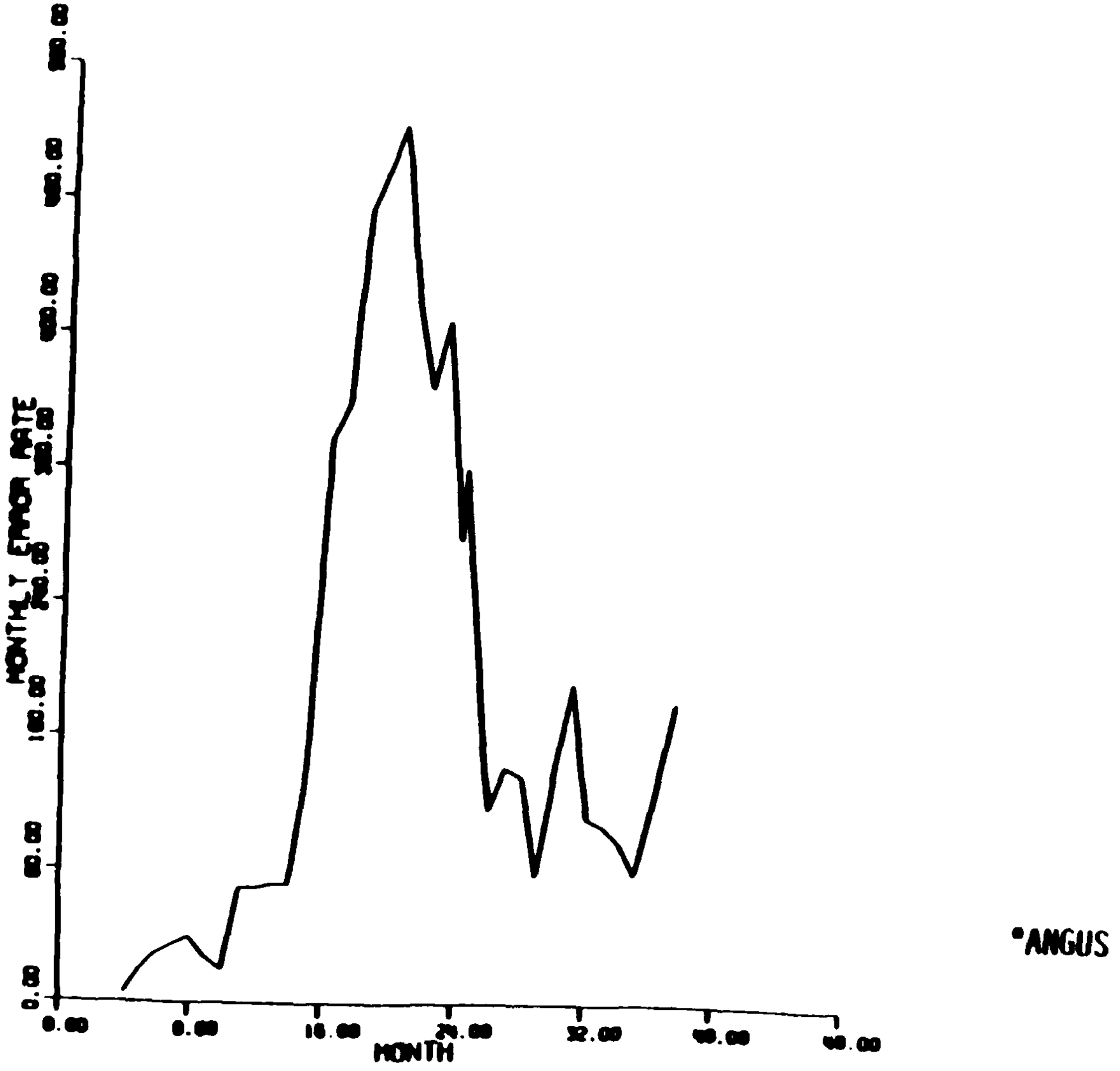


FIGURE 3-8 MONTHLY ERROR DETECTION RATE {ANGU79}

there are considerable difficulties in obtaining credible figures on this, particularly where parts of the test were conducted by the developer and other parts by the Government or a separate contractor. Also, because test is the project activity most likely to be under budget and schedule pressure, substantial parts of test are sometimes conducted as a supplemental project for which data are not recorded in the main project records.

A second alternative is the total calendar time devoted to test for use as a comparison among projects of approximately equal size. Normalization by dividing by total lines of code may be inappropriate because of nonlinearities affecting large projects. However, normalized calendar time will be evaluated as a metric for the amount of test during this study.

As a third alternative, the number of separate test teams involved will be evaluated. In a major project, the following may be responsible for major phases of software test:

- Software Developer,
- Developer's Software Test or QA Staff,
- System Integrator,
- Independent Validation Contractor.
- Air Force Test Agent (Air Force Operational Test and Evaluation Command),
- Sponsor (Air Force Systems Command), and
- End User (Air Force Operational Command).

The more teams involved, the more thoroughly the system will be tested. The metric, TE, will be examined in these three forms during the validation phase of the project and the form which exhibits the best results will be chosen. The three forms are:

(1) TE - 40/AT

where AT - the percent of the development effort devoted to testing.

(2) - 40/AT

where AT - the percent of the development schedule devoted to testing.

$$(3) \quad - \sum_{i}^{n} TT(i)$$

where TT is a factor (to be determined by regression) associated with each test team mentioned above and n is the number of test teams applied.

- Test Methodology (TM)

The test methodology used is another element by which to assess the thoroughness of testing. One measure, TM, that suggests itself is the use of test tools and testing techniques. In most cases the tools are being operated by a staff of specialists who are also aware of other advances in software test technology. The primary emphasis will be on classifying the test environment by the tools and techniques used. Distinctions based on the type of test tools and techniques used will be made.

A technique and handbook for doing this assessment (or classification) has been developed. In the Software Test Handbook [PRES84], a technique to determine what tools and techniques should be applied to a specific application is provided. That technique is illustrated in Figure 3-9 and results in a recommended set of testing techniques and tools. Our approach will be to use that recommendation to evaluats the techniques and tools applied on a particular development. This evaluation will result in a score that will be the basis for this metric as follows:

$$TM - k_t * TR/TU$$

where TU is the number of tools and techniques used and TR is the number recommended. k_t is a constant determined by regression.

The tool and technique checklist in [PRES84] is specifically to be used to assess testing. The tool and technique checklist shown earlier (Table 3-7) was for the development phases of requirements, design, and coding.

- Test Coverage (TC)

This metric assesses how thoroughly the software has been exercised during testing. If all of the code has been exercised then there is some level of confidence established that the code will operate reliably during operation. Typically however, test programs do not maintain this type of information and a significant portion (up to 40%) of the software (especially error handling code) is never tested. Tools such as JAVS, FAVS, and CAVS

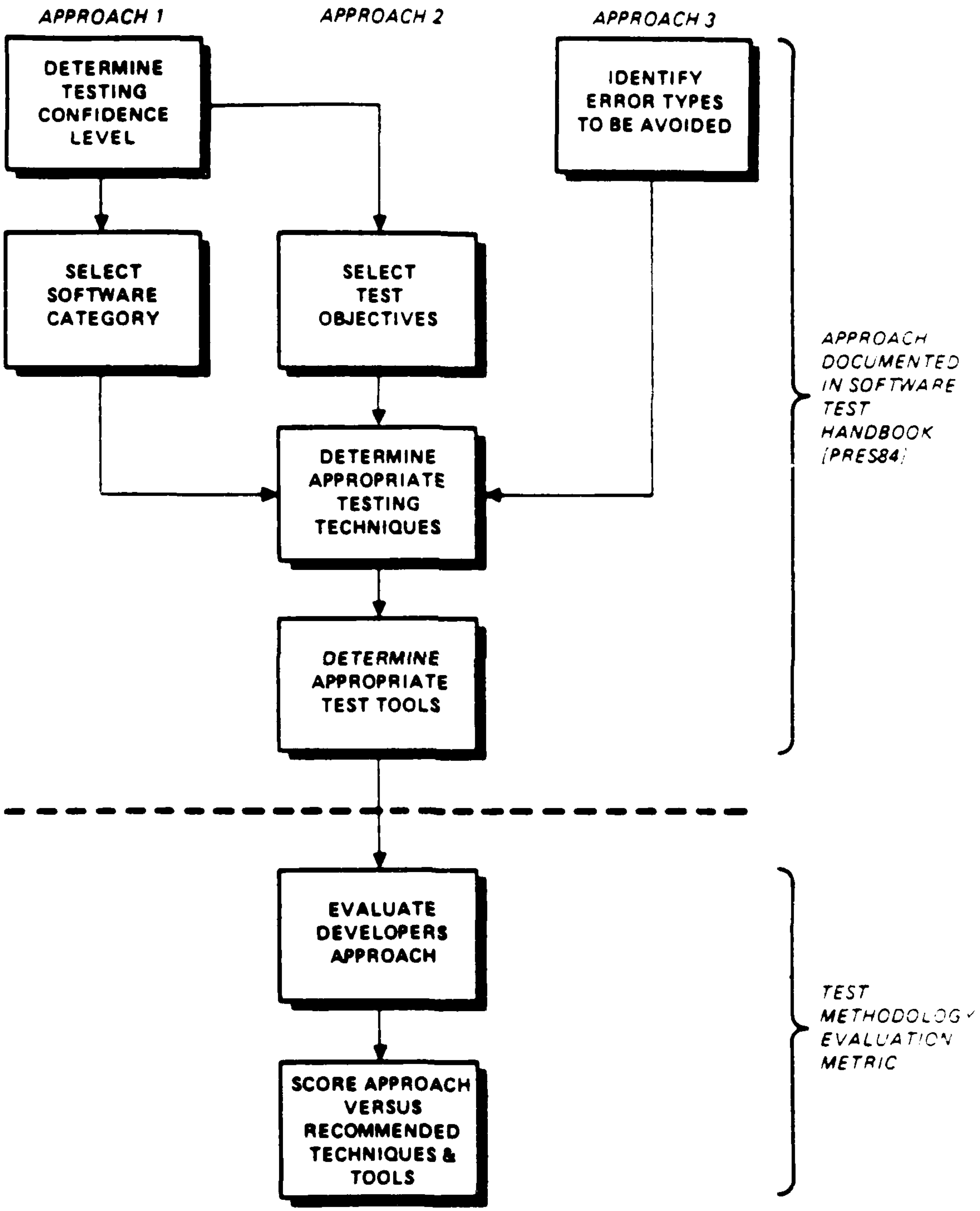

APPROACH 1
APPROACH 2
APPROACH 3
DETERMINE TESTING CONFIDENCE LEVEL
IDENTIFY ERROR TYPES TO BE AVOIDED
SELECT SOFTWARE CATEGORY
SELECT TEST OBJECTIVES
DETERMINE APPROPRIATE TESTING TECHNIQUES
DETERMINE APPROPRIATE TEST TOOLS
APPROACH DOCUMENTED IN SOFTWARE TEST HANDBOOK [PRES84]
EVALUATE DEVELOPERS APPROACH
SCORE APPROACH VERSUS RECOMMENDED TECHNIQUES & TOOLS
TEST METHODOLOGY EVALUATION METRIC

FIGURE 3-9. TEST METHODOLOGY ASSESSMENT APPROACH

(developed under RADC contracts) provide such information.

This metric could be calculated in three ways depending on the phase of testing as follows:

$$TC - k_{tc} * 1/VS$$

where k_{tc} is a constant determined by regression

VS - VS1 during unit testing
- VS2 during integration testing
- VS3 during system testing

and

VS1 - (PT/TP + IT/TI)/2
where PT - execution branches tested
TP - total execution branches
IT - input tested
TI - total number of inputs

VS2 - (MT/TM + CT/TC)/2
MT - units tested
TM - total number of units
CT - interfaces tested
TC - total number of interfaces

VS3 - RT/NR
RT - Requirements tested
NR - total number of requirements

3.3.2.3 Operating Environment (E)

Several characteristics of the operational environment, experienced during OT&E, should be accounted for in estimating reliability. Again, during OT&E we are trying to extrapolate the observed failure rate (F) into operations. The characteristics we want to account for are the workload and the variability of inputs. These two characteristics, for which we have developed metrics, represent the stress of the operational environment on the software. The metrics will be multipliers which will raise or lower the estimated failure rate depending on the degree of stress (E - EW * EV).

- **Workload (EW)**

  The relationship between the workload and software failure rate has been investigated at Stanford University and a very significant positive correlation has been reported [ROSS82]. The basic concept underlying this phenomena is that more unusual situations (program swapped in and out of memory, queued I/O, wait states, etc.) are encountered in a heavy workload, and the application programmer may not have anticipated all the situations. In addition, system software will tend to fail more often when used more often.

The measured workload will be transformed into a stress metric as follows:

$$EW = k_{ew} * ET/(ET-OS)$$

where OS is the amount of Operating System overhead used, ET is the total execution time, k_{ew} is a constant determined by regression. This form of relationship (linear) will be developed if applicable. If not a more general relationship, EW = f (OS), will be developed.
The use of operating system overhead was chosen because it is usually available. Other alternatives are number of system calls per minute, number of paging requests, and number of I/O operations.

- Variability of Input (EV)

Variability of the input is the primary determinant of software reliability in some models, such as the ones proposed by Nelson and Lipow [DACS79] and Roger Cheung [CHEU81]. The basic concept here is that the greater the variability of inputs to the program the more like'y an unanticipated input will be encountered and the program will fail. Neither one of these models is supported by sufficient data to permit direct evaluation of the effect of variability on failure frequency, however. Nelson and Lipow proposed partitioning of the input data set, and an index of variability can then be derived from the number of partitions accessed during one time period or one run. This appears practical in only a very limited number of applications. Cheung uses the calling sequence as an indicator of variability, a somewhat more easily implemented measure, but still targeted primarily to a research environment. It is proposed to use the frequency of exception conditions as a practical measure of variability in the current effort. The monitoring of exception conditions is accomplished by hardware provisions which are incorporated in many current computers. Significant correlation between the frequency of exception conditions and failure rate has been demonstrated [IYER83].

The metric will be:

$$EV = .1 + 4.5EC$$

where EC is the number of exception conditions encountered per hour.

The constant value of .1 and the coefficient of 4.5 where derived as a result of the analysis in [IYER83].

3.4 TIMING OF METRIC APPLICATION DURING THE LIFE CYCLE

Figure 3-10 indicates when during the development phase each of the metrics identified would be applied. This application requires data collection, described in the next section, and then use in the prediction or estimation procedures described in Volume II.

METRICS	Concept Development/ Acquisition Initiation	Mission/ System/ Software Definition	Software Requirements Analysis	Preliminary and Detailed Design	Coding and Units Testing	CSC Integration and Testing	CSCI Testing	System Integration and Testing	Operational Test and Evaluation	Production and Deployment
■ Application Type	●	●								
■ Development Environment			●							
■ Software Characteristics										
□ Anomaly Management				●						● *
□ Traceability				●	●					●
□ Quality Review Results				●						●
□ Language Type					●					●
□ Program Size					●					●
□ Modularity					●					●
□ Extent of Reuse					●					●
□ Complexity					●					●
□ Standards Review Results					●					●
■ Failure Rate During Testing						●	●	●	●	
□ Test Effort						●	●	●	●	
□ Test Methodology						●	●	●	●	
□ Test Coverage						●	●	●	●	
□ Workload						●	●	●	●	●
□ Input Variability						●	●	●	●	●

FIGURE 3-10. TIMING OF METRIC APPLICATION

*During Production and Deployment, modifications to the Software should be tracked by reapplication of the metrics

4.0 DATA COLLECTION IN SUPPORT OF THE SOFTWARE RELIABILITY PREDICTION AND ESTIMATION METHODOLOGY

4.1 DATA COLLECTION APPROACH

One of the more significant undertakings of this project was the data collection activities associated with demonstrating and validating the methodology. The goals during this phase of the project were:

- Filter the candidate measurements, ie eliminate measurements that had no potential for utility in the methodology and identify those that appear to have predictive or estimation potential.

- Establish a data base from which a draft handbook (Volume II) could be developed.

- Collect a set of data with which preliminary validation efforts could be performed. These validation efforts are preliminary because as a result of them some changes to the measurements have been made (thus requiring further iteration) and because a more exhaustive set of data would be required to perform more extensive validation.

- Establish data collection procedures for the Reliability Prediction and Estimation Methodology.

The overall approach to the data collection is illustrated in Figure 4-1.

During Phase I, a number of projects were identified as potential sources of data for this project. Also during Phase I, a literature search was conducted. This literature search had three purposes. One was to identify reliability measures that had been established and tried within the industry. A second was to further extend the references available to software reliability practioners and document terminology (see Appendix A). The third reason was to collect any documented experiences as part of the data base to be used in this project. The RADC Data and Analysis Center for Software (DACS) and the NASA Software Engineering Laboratory (SEL) data bases were also utilized.

Each software project, data base, and reference were analyzed for applicability to this effort. The analysis mainly consisted of identifying whether enough documentation, source code, and failure history existed and was available for use. If this data existed and was available, further investigations were conducted to determine where in the life cycle the data was from, how reliable the data was, and how current the data was. Some projects and sources were eliminated from consideration because of these factors. The resulting set comprised the candidate set

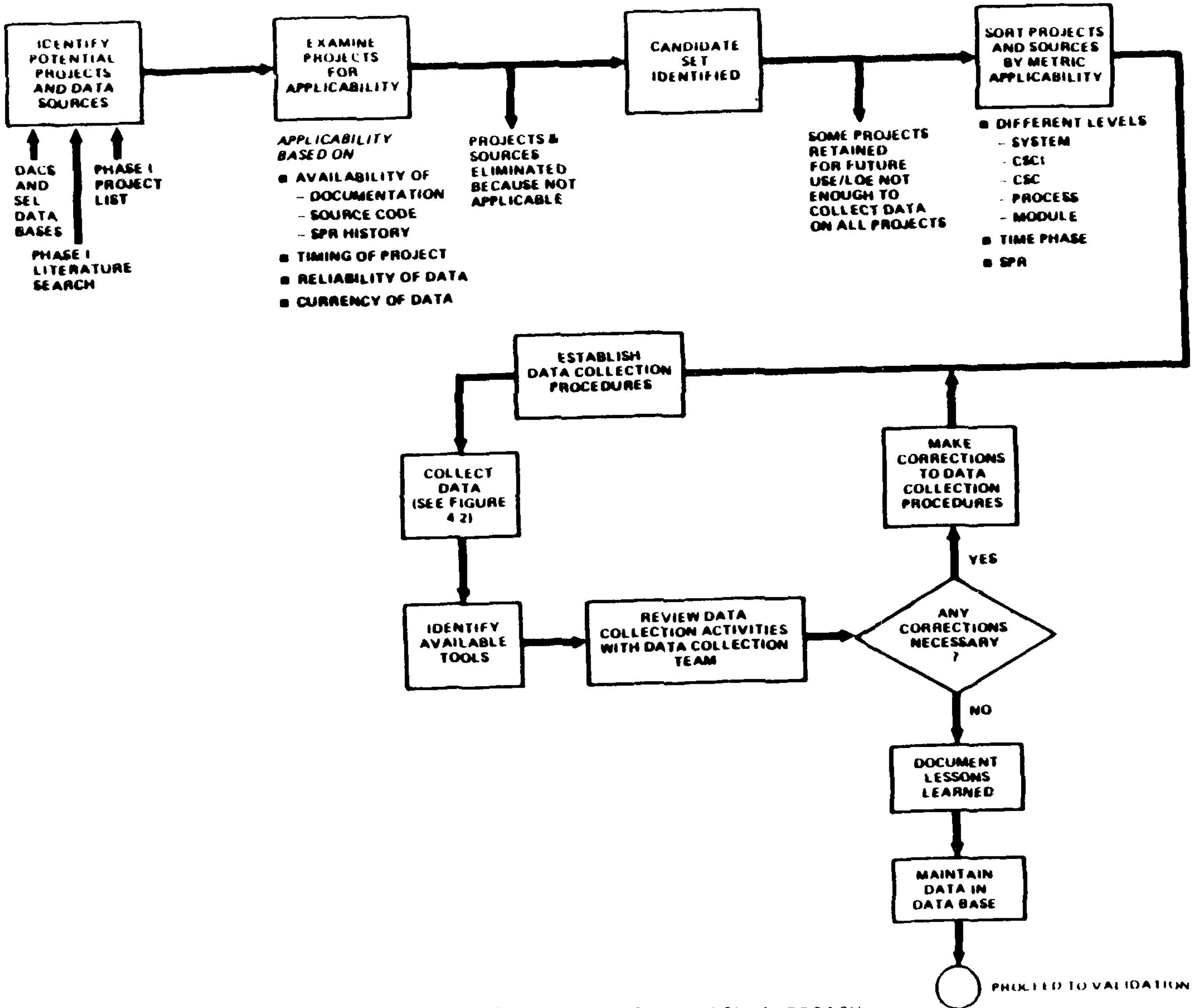

IDENTIFY POTENTIAL PROJECTS AND DATA SOURCES
DACS AND SEL DATA BASES
PHASE I LITERATURE SEARCH
PHASE I PROJECT LIST
EXAMINE PROJECTS FOR APPLICABILITY
APPLICABILITY BASED ON
■ AVAILABILITY OF
– DOCUMENTATION
– SOURCE CODE
– SPR HISTORY
■ TIMING OF PROJECT
■ RELIABILITY OF DATA
■ CURRENCY OF DATA
PROJECTS & SOURCES ELIMINATED BECAUSE NOT APPLICABLE
CANDIDATE SET IDENTIFIED
SOME PROJECTS RETAINED FOR FUTURE USE/LOE NOT ENOUGH TO COLLECT DATA ON ALL PROJECTS
SORT PROJECTS AND SOURCES BY METRIC APPLICABILITY
■ DIFFERENT LEVELS
- SYSTEM
- CSCI
- CSC
- PROCESS
- MODULE
■ TIME PHASE
■ SPR
ESTABLISH DATA COLLECTION PROCEDURES
COLLECT DATA (SEE FIGURE 4 2)
IDENTIFY AVAILABLE TOOLS
REVIEW DATA COLLECTION ACTIVITIES WITH DATA COLLECTION TEAM
ANY CORRECTIONS NECESSARY ?
YES
MAKE CORRECTIONS TO DATA COLLECTION PROCEDURES
NO
DOCUMENT LESSONS LEARNED
MAINTAIN DATA IN DATA BASE
PROCEED TO VALIDATION

FIGURE 4-1 DATA COLLECTION APPROACH

of projects and data sources. As many as possible were included in the data collection and validation activities. A few were not because the level of effort of this project prohibited their inclusion. Those projects have been retained for future analysis. The next paragraph, 4.2, identifies all of the candidate projects and data sources.

The next step in the data collection approach was to sort the projects and data sources as to their applicability to the candidate prediction and estimation measurements identified in the preceding section. This sort was necessary for two reasons. The first is that the measurements themselves represent different levels of data spanning system level characterizations down to module level measures, different time periods in a system life cycle, and require different levels of problem reporting association. Thus the measurements require different levels of detail and this step provided for the process of aligning projects and data sources with metrics. A second reason this step was necessary was that all the projects and data sources were not compatible in terms of data availability. Some only provided data at a system level. Some only provided detailed data for certain measurements and not all. This non-homogeneity is a fact of life, all data collection efforts are faced with it. Our approach to dealing with this fact was to gather enough data from enough sources to be able to fully cover all of the measurements. There is further discussion of this point in paragraph 4.2.

Data collection procedures were established and the data collection activities proceeded. Periodic data collection team meetings were held to not only check progress, but to discuss problems being encountered so that corrective actions could be taken. As a result of these meetings a number of lessons-learned have been recorded and are discussed in paragraph 4.4. As part of the data collection activities, any tools that would aid in the data collection were identified and used. The tools used are described in paragraph 4.3.

Figure 4-2 is a more detailed illustration of the data collection activities. Two RADC Technical Reports (RADC TR 85-37 and RADC TR 84-53) were key to the data collection activities. RADC TR 85-37 provided a set of worksheets associated with many of the Software Characteristics Metric (Anomaly Management, Traceability, Quality Review Results, Size, Modularity, Complexity, and Standards Review Results). RADC TR 84-53 provided a process for evaluating the Testing Methodology. The data collection activities essentially paralleled an actual application of the Reliability Prediction and Estimation Methodology (see Volume II). A set of data collection tasks was oriented toward collecting the data associated with the prediction metrics. This set was generally applied to the documentation and source code. Another set was oriented toward collecting the data associated with the estimation metrics. This set was generally applied to the test (in some cases operational) results. As part of this second set, failure data was collected which later was used to demonstrate

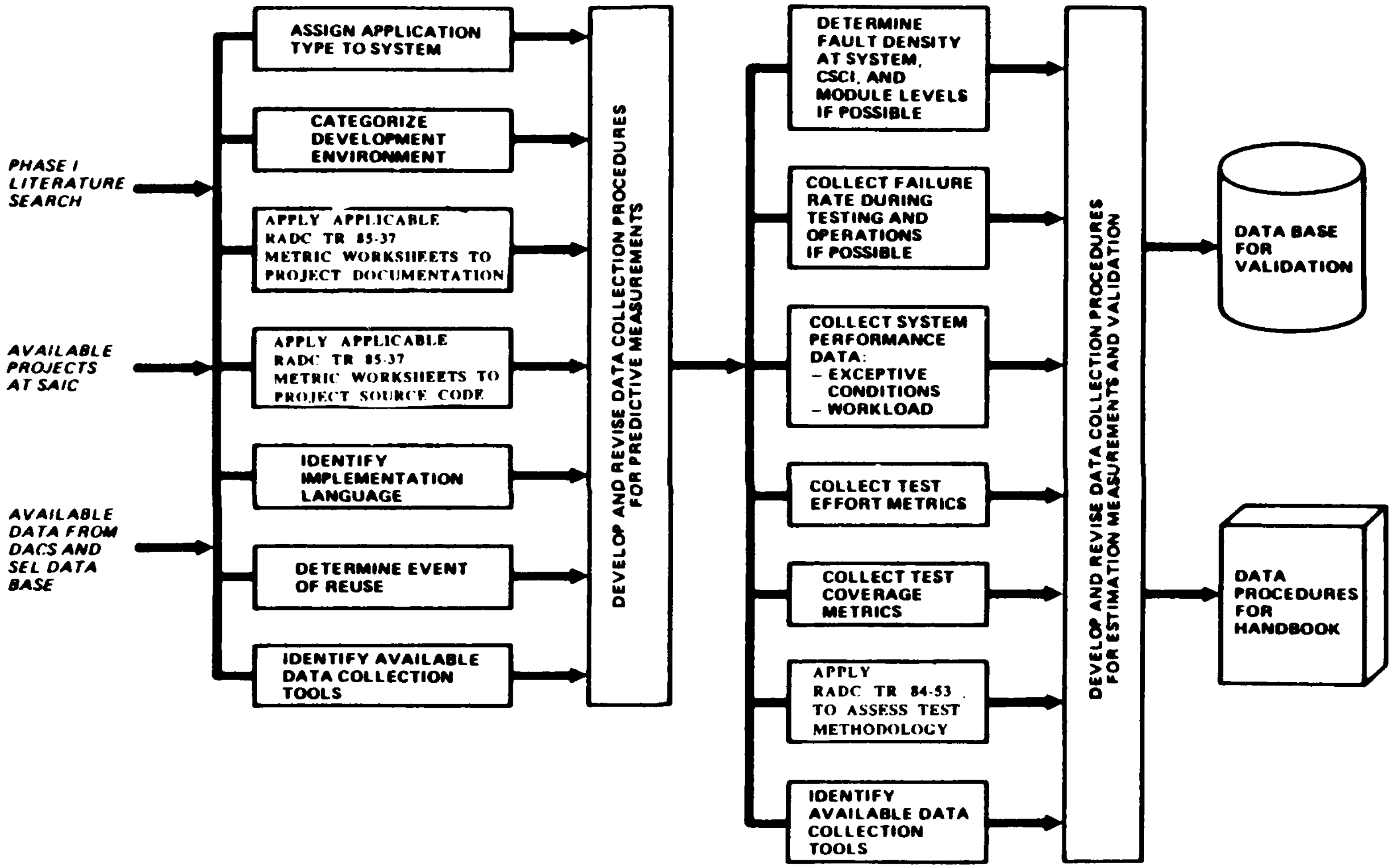

PHASE I LITERATURE SEARCH
AVAILABLE PROJECTS AT SAIC
AVAILABLE DATA FROM DACS AND SEL DATA BASE
ASSIGN APPLICATION TYPE TO SYSTEM
CATEGORIZE DEVELOPMENT ENVIRONMENT
APPLY APPLICABLE RADC TR 85-37 METRIC WORKSHEETS TO PROJECT DOCUMENTATION
APPLY APPLICABLE RADC TR 85-37 METRIC WORKSHEETS TO PROJECT SOURCE CODE
IDENTIFY IMPLEMENTATION LANGUAGE
DETERMINE EVENT OF REUSE
IDENTIFY AVAILABLE DATA COLLECTION TOOLS
DEVELOP AND REVISE DATA COLLECTION PROCEDURES FOR PREDICTIVE MEASUREMENTS
DETERMINE FAULT DENSITY AT SYSTEM, CSCI, AND MODULE LEVELS IF POSSIBLE
COLLECT FAILURE RATE DURING TESTING AND OPERATIONS IF POSSIBLE
COLLECT SYSTEM PERFORMANCE DATA:
– EXCEPTIVE CONDITIONS
– WORKLOAD
COLLECT TEST EFFORT METRICS
COLLECT TEST COVERAGE METRICS
APPLY RADC TR 84-53 TO ASSESS TEST METHODOLOGY
IDENTIFY AVAILABLE DATA COLLECTION TOOLS
DEVELOP AND REVISE DATA COLLECTION PROCEDURES FOR ESTIMATION MEASUREMENTS AND VALIDATION
DATA BASE FOR VALIDATION
DATA PROCEDURES FOR HANDBOOK

FIGURE 4-2 DATA COLLECTION ACTIVITIES

and validate the use of the measurements as predictors and estimators of software reliability.

In both cases, an initial set of data collection procedures were produced to aid in the data collection activities and based on the experience revised. The data collection procedures are included as Appendix B to Volume II.

The primary end result of the data collection activities, besides the data collection procedures, was a data base that could be used to demonstrate and validate the measurements identified in Section 3 of this report.

4.2 DATA SOURCES

The sources of data for this project fall into three categories: existing data bases such as the DACS and SEL data bases; results and data reported in the literature; and data collected from projects during this contract effort.

In the following paragraphs, a brief description of each source of data used during this effort is described and a reference, if appropriate, is sited. The type of data available from each of these projects is also described. In situations where the project sited was used as a source for detailed data, the various documents and data available is identified. A summarization of these data sources is in Table 4-1.

Radar Control System (1)

This project's error history was documented in [WILL77] and compared with other projects in [FISH79]. It is a real-time control system for a land-based radar complex. It was written in JOVIAL and assembly language. The data available was primarily used to distinguish fault densities by application type. The failure data represented integration and operational test results.

Avionics Control System (2)

This project's error history was documented in [FRIE77] and compared with other systems in [FISH79]. It is an avionics control system that was developed in JOVIAL and assembly language. The data available was primarily used to distinguish fault densities by application type. The failure data represented module verification, intermodule compatibility.

Satellite Command and Control System (3)

This project's error history was documented in [THAY76] and compared with other systems in [FISH79]. It is a large command and control system written in JOVIAL and assembly language. The data available was primarily used to distinguish fault densities by application type. The failure data represented development

TABLE 4-1 DATA SOURCES

NO	DATA SOURCE	DESCRIPTION	REF.	APPLICABLE MEASUREMENTS																		
				FAULT DENSITY	FAILURE RATE	ERROR CATEGORIZATION	APPLICATION	DEVELOPMENT ENVIRONMENT	ANOMALY MGMT	TRACEABILITY	QUALITY REVIEW	LANGUAGE	SIZE	MODULARITY	REUSE	COMPLEXITY	STANDARDS REVIEW	TEST EFFORT	TEST METHODOLOGY	TEST COVERAGE	WORKLOAD	INPUT VARIABILITY
1	RADAR CONTROL SYSTEM	REAL TIME CONTROL SYSTEM FOR RADAR COMPLEX	[WILL77], [HEIN79]	✓		✓	✓	✓				✓	✓	✓		✓			✓			
2	AVIONICS CONTROL SYSTEM	AVIONICS CONTROL SYSTEM ON BOARD AIRCRAFT	[PRIE77], [FRIN79]	✓		✓	✓	✓				✓	✓						✓			
3	SATELLITE C^2 SYSTEM	GROUND BASED C^2 SYSTEM	[THAY76], [FRIN79]	✓		✓	✓	✓				✓	✓									
4	ABM C^2 SYSTEM	GROUND BASED C^2 SYSTEM FOR ANTI BALLISTIC MISSILE	[BASI77], [FRIN79] [MOTL76]	✓	✓	✓	✓	✓				✓	✓					✓	✓			
5	C^3I SYSTEM	CLASSIFIED REAL TIME C^3I SYSTEM	N/A	✓	✓		✓						✓	✓								
6	INTERACTIVE SYSTEM	INTERACTIVE IN HOUSE SYSTEM	N/A		✓		✓															
7	SCIENTIFIC SYSTEM	SCIENTIFIC BATCH PROCESS	[HECH77]		✓		✓															
8	FLIGHT CONTROL	FLIGHT CONTROL FOR ADV FIGHTER TECH INTEGRATION	[MACK83]		✓		✓	✓											✓			
9	C^2 OPERATING SYSTEM	CLASSIFIED REAL TIME OPERATING SYSTEM	N/A		✓		✓															
10	US ARMY NATIONAL TRAINING CENTER	INTERACTIVE GRAPHICS DISPLAY REAL TIME MESSAGE HANDLING, COMMAND AND CONTROL	SAIC PROJECT DATA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
11	ALCM	MISSION PLANNING	SAIC PROJECT DATA	✓			✓					✓	✓	✓		✓						
12	DIGITAL FLIGHT CONTROL SYSTEMS	DIGITAL FLIGHT CONTROL SYSTEMS 2 SOURCES OF DATA	[HECH83] [ROCK81]	✓			✓					✓	✓									
13	INTERACTIVE SYSTEMS	INTERACTIVE MILITARY SYSTEMS, 6 DIFFERENT SYSTEMS	[MUSA79]	✓	✓		✓															
14	ELECTRONIC SWITCHING SYSTEM	COMMUNICATIONS BELL LABORATORIES	[DAVI81]		✓		✓	✓														
15	SCIENTIFIC	VIKING FAILURE DATA FROM SCIENTIFIC SOFTWARE	[MAXW78]	✓			✓															
16	SATELLITE C^2	CLASSIFIED C^2	N/A	✓			✓	✓				✓	✓	✓		✓	✓	✓				
17	EMERGENCY RESPONSE SYSTEM	PROCESS MONITORING		✓			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
18	SUPPORT	DATA REDUCTION SYSTEM	[WAGO73]		✓		✓															
19	COMMAND AND CONTROL	REAL TIME DISPLAY MANAGEMENT AND COMMAND EXECUTION SYSTEMS	[MUSA79]	✓	✓		✓						✓									
20	INTERACTIVE OS	COMPUTER CENTER OPERATING SYSTEMS	[VERM81]		✓		✓															
21	IMAGE PROCESSING	DATA MANAGEMENT, IMAGE PROCESSING,DATA RECORDING	[GRAB82]	✓			✓	✓				✓	✓	✓		✓						
22	ALCM	MISSION DATA PREP DIGITAL FLIGHT CONTROL	[HECH83] [PRIS81]	✓			✓					✓	✓									
23	FLIGHT CONTROL	FLIGHT CONTROL	[PRIS81] [HECH83]	✓			✓					✓	✓									
24	SUPPORT	SUPPORT PROGRAMMING	[PRIS81] [HECH83]	✓			✓					✓	✓									
25	C^2	TELEMETRY PROCESSING SATELLITE C^2	[HECH83] [BASI77] [CARD82] [TURN81]	✓			✓					✓	✓									
26	MIS	SMALL BUSINESS MIS	[HEBB86]	✓			✓	✓				✓	✓					✓				
27	ARGUS	TELEMETRY PROCESSING	[TROY86]		✓		✓															
28	INTERACTIVE SYSTEM	ON LINE DATA PROCESSOR	[MEYA]		✓		✓															
29	SIGNAL PROCESSING	SIGNAL PROCESSING SYSTEM APPLICATIONS	[MONT79]	✓	✓	✓	✓						✓									
30	MIS	LOGISTICS	[ASHM83]	✓			✓					✓	✓	✓								
31	SIMULATION	COMPUTER SIMULATION	[WHIT78]	✓		✓	✓						✓									
32	C2	GROUND BASED C2 SYSTEM	[THA478]	✓		✓	✓					✓	✓	✓								
33	SIMULATION	REAL TIME SIMULATOR	[THA478]	✓		✓	✓					✓	✓									

testing, validation testing, acceptance testing, integration testing and operational testing results.

ABM Command and Control System (4)

This project's error history was documented in [BAKE77] and [MOTL76]. It was compared with other systems in [FISH79]. It is a ground-based command and control system for an anti-ballistic missile system. It was written in the CENTRAN programming language and the failure data collected represented unit testing, functional testing and system integration testing results. The data available was used primarily to distinguish fault densities among applications.

C^3I System (5)

This project is a classified Command, Control, Communications and Intelligence system. Due to the classification, the system is not identified nor is documentation available. The failure history, collected during an operational window of four months was provided in an unclassified form for use in this effort. The data available consists of failure rate data.

Interactive System (6)

This project is an interactive system developed by a Government fiscal agency for use internally. The data represents operational failures during a six month period during 1981. The data available was used primarily to distinguish failure rates by application type.

Scientific System (7)

This project is the Launch Support Data Base (LSDB) program at Vandenberg AFB. The failure data was derived and reported in [HECH77]. The data represents failure rate data collected during development and integration testing prior to acceptance. It was used primarily to distinguish failure rates by application type.

Flight Control System (8)

This project is the digital flight control system of the Advanced Fighter Technology Integration (AFTI) F-16 program. The failure rate observed during flight testing over a 13 month period was reported in [MACK83a,b].

Command and Control Operating System (9)

This project is a classified ground-based command and control system. The software problem reports reported over a 25 month period were collected. The average amount of testing done per month was 200 hours.

Training System (10)

This project is a large complex training system built to support the U.S. Army. The system is comprised of a real-time message handling subsystem, interactive graphics workstations, and post-operations play back. The system provides real-time display of instrumented exercises to observers. This project was used as a source of most of the detailed data required. A complete set of development documentation as well as source code, test results and operational performance data was available or collected for analysis.

Mission Planning System (11)

The mission planning system for the Air Launch Cruise Missile was a source of Independent Verification and Validation problem reports. Development problem report statistics were available for an initial version of the system. This system contains planning software and report generation software.

Flight Control System (12)

This data set contains data from four flight control and related program applications. The data is reported in [PRES81] and [ROCK81] and analyzed in [HECH83]. The data reported is fault density and was used primarily for establishing the application type.

Interactive System (13)

This data set represents four interactive systems, one a commercial system and three military systems. These data sets are reported in [MUSA79] (as systems 5, 17, 27, and 40). Each system is a large interactive system and the fault density data provided is from system test.

Electronic Switching System (14)

The source for this data set is [DAVI81]. It is an electronic switching system developed by Bell Laboratories. The data presented is from installation and operations, for the system and represents a very high reliability.

Scientific System (15)

This data set is from the Viking project at the Jet Propulsion Laboratory [MAXW78]. Failure rate data is provided from a four month period during operations.

Ground-Based Command & Control (16)

This data set is from a classified command and control system. The data available are fault density and source code characteristics. The failure data is from development and integration testing.

Process Monitoring System (17)

This data set is from an Emergency Response Information System developed to monitor a Nuclear Power Plant. Data available includes fault density, development documentation, source code, and code characteristics. The failure data available represents problems recorded during acceptance testing and operational use.

Support System (18)

This data set is a data reduction system developed for in-house use on the F-11D project [WAGO73]. Failure rate data is available. The data was used primarily to determine Application Type baselines.

Command and Control Systems (19)

This data set is comprised of four real-time display management and command execution systems, all command and control applications. The data, consisting of fault density and failure rate data, is recorded in [MUSA79] as systems 1, 2, 3 and 4. This data was used primarily to establish Application Type baselines.

Interactive Operating System (20)

This data set represents failure rates for two computer installations at Stanford University [IYER81]. The data spans three years of operational use. This data was used primarily to establish Application Type baselines.

Image Processing System (21)

This data set was reported in [GRAS82] for an Image Processing System development. During the development, a committment to collect software quality metrics was made. The results of this application are reported in the above reference. Failure data was collected during two incremental builds of a system and during acceptance testing.

Flight Control (22)

This data set is for the ALCM Operational Flight System reported in [HECH83]. Fault density data is available and was used primarily to establish an Application Type baseline.

Flight Control (23)

This data is also in [HECH83] and represents several projects or generations of the same system. Fault Density was available.

Support Programs (24)

This data represents support software and a simulator supporting flight control software development and testing. It is summarized in [HECH83]. This study used the summarization of the fault density experience data to help establish an Application Type baseline.

Satellite C^2 (25)

This data is a subset of data available from the SEL data base. It is reported in [HECH83], [BASI77], [CARD82] and [TURN81]. The fault densities recorded for 11 different projects or software systems were used to help establish an Application Type baseline. All of the systems were related to the Satellite C^2/Telemetry processing systems developed and operated at NASA/Goddard.

MIS (26)

This data was reported in [HIER86]. It is from four projects involving small business systems. An analysis of the inpact and benefit software quality metrics can have was reported in the reference. The development environment and test effort was available as well as fault density for these four projects ranging in size between 10,000 and 30,000 lines of code.

ARGOS (27)

This data was reported in [TROY86] as a study of software failure reporting within a large data processing center. The data processing center is for the purpose of acquiring, processing and distributing telemetry data. Failure rate information is provided.

Interactive System (28)

This project involved a dual CPU processing system able to handle 500 on-line users [MIYA-]. Software as well as hardware reliability goals were set for project and progress toward the achievement of these goals was monitored. An evaluation of reliability models [GOEL83] was made. Failure rate data was provided.

Signal Processing (29)

Failure rate and failure density data is provided in [MEND79] for two signal processing applications. Additionally an evaluation of error types and validity of reliability models are presented.

MIS (30)

Fault density data is provided based on an evaluation of an Army Logistics Support MIS system [LEHM82]. Over 1.6 million lines of code are represented in the study.

Simulation (31)

Fault density and error categorization data is presented in [WEIS78] for a computer architecture simulation facility.

C2 System (32)

This data source is project 2 reported in [THAY76]. Fault Density and software and error characteristics are provided for this command and control system written in JOVIAL.

Simulation (33)

This data source is project 5 reported in [THAY76]. It is a simulator developed in FORTRAN and Assembly language. Fault Density and software and error characteristics are provided.

Thirty-three (33) data sources are identified representing 59 different projects. Most of these data sets were used during this project to establish some baseline reliability numbers for different types of applications. Several were used to evaluate the candidate predictive and estimation measures identified in the preceeding section. Data Sources 10 and 17 specifically were projects from which detailed data were collected for the purpose of demonstrating and validating. The DACS and SEL data bases were utilized to the extent possible. Data Sources 1, 2, 3, 4, 13, 19, 25 are in either the DACS or SEL. This data was typically analyzed and reported elsewhere (references are noted).

4.3 EXAMPLE DATA

The data collected for this study basically is that set of data required to calculate the metrics described in Section 3. A complete set was delivered to RADC as part of this contract. To illustrate the data collected, examples are provided in this section. The data is presented by metric here to facilitate reference and correlation to the validation results presented in the next section.

4.3.1 Application

Table 4-1 provided a brief description of each data source with respect to the type of system (application type) represented by the data source. Table 4-2 presents a summary of the fault density or failure rate data collected for each of these data sources.

The fault density depicted is the number of failures (software problems reported) divided by the number of executable source lines of code which make up the software system.

In most cases, collecting this data was straight forward. Data bases examined or articles referenced typically identified the

TABLE 4 2 SUMMARY OF FAULT DENSITY/FAILURE RATE

Application	Data Source	Number of Systems	Fault Density	Failure Rate (Computer Operation Hours)			Comments
				Avg Test	End Test	Operation	
AIRBORNE/RT	2	1	.017				
AIRBORNE/RT	8	1		.08			
AIRBORNE/RT	12	2	.0086				
			.0018				
AIRBORNE/RT	22	1	.0029				
AIRBORNE/RT	23	3	011				
			.027				
			.021				
STRATEGIC/RT	1	1	.016	.46*	.075*		*for 2 builds
STRATEGIC/RT	3	1	.039				
STRATEGIC/RT	4	2	.019				
			.017				
STRATEGIC/RT	6	1	.0085			.0034	
STRATEGIC/RT	9	1		.064	.02		
STRATEGIC/RT	14	1				.0007	
STRATEGIC/RT	16	1	.005				
STRATEGIC/RT	26	11	.0042				
			.0007				
			.0001				
			.0005				
			.0010				
			.0016				
			.0014				
			.0008				
			.0029				
			.0001				
			.001				
STRATEGIC/RT	27	1*				0032	*3 subsystems reported
						019	
						028	
STRATEGIC/RT	28	1			.0025		
STRATEGIC/RT	29	2	004	66	.016		
			0019	17	.019		
STRATEGIC/RT	32	1	015				
STRATEGIC/B	11	1	054*				*includes IVV PRs

TABLE 4.2 SUMMARY OF FAULT DENSITY/FAILURE RATE (CONT.)

Application	Data Source	Number of Systems	Fault Density	Failure Rate (Computer Operation Hours)			Comments
				Avg. Test	End Test	Operation	
TACTICAL/RT	10	1	.016	1.04	.63	.18	
TACTICAL/RT	19	4	.0125	6.4	1.1	.1	
			.004	1.7	.54	.05	
			.0032	1.8	.57	.066	
			.0031	2.9	.36	.145	
PROCESS CONTROL/RT	17	1	.002				
PROCESS CONTROL/OL	21	1	.0016			.007	
PRODUCTION/OL	6	1				.72	
PRODUCTION/OL	7	1	.035	68	9		
PRODUCTION/OL	13	4	.0013		.13		
			.0025		.125		
			.0013		.0049		
			.0022		.0062		
PRODUCTION/B	20	2	.01			.024	
			.015			.017	
PRODUCTION/B	26	4	.0075				
			.007				
			.0095				
			.01				
PRODUCTION/B	30	1	.0012				
PRODUCTION/B	15	1				.03	
DEVELOPMENTAL/RT	33	1	.024				
DEVELOPMENTAL/S	18	1		170	20.9		
DEVELOPMENTAL/S	24	2	.0093				
			.0019				
DEVELOPMENTAL/S	31	1	.014				

number of failures recorded against a system and also the size of the system. In some cases, the failures (problem reports) and size data were provided by module or subsystem and had to be totalled.

The failure rates depicted are the average failure rate experienced during testing of the system, i.e., the number of failures observed divided by the total time spent testing, the failure rate observed at the end of the test phase, and the failure rate observed during operation of the system. The failure rate at end of test is calculated by taking the average failure rate observed during the last three test periods. Computer operational time is used. This table has been organized by Application Type. An analysis of this data is presented in Section 5. CPU execution time could be used but since it was rarely available, computer operation time is used as a close approximation of CPU execution time. Where available, a conversion factor is used to translate CPU execution time to computer operational time.

Software failure rate data is typically more difficult to find reported or to have collected. The missing element is usually the time. At a minimum problem reports should be dated or operator's logs annotated when problems are encountered. Figure 4-3 is an example where the problem report history (data source 9) is time stamped only by month. In this case (data source 9 is a classified real time system), this is the only data available from this project except an estimate that on the average 200 hours of computer time was spent testing the software each month. This data is enough to calculate the failure rate shown in Table 4-2.

Although Table 4-2 is at present only partially populated, the trends within the columns are about as expected. This is particularly true for the end of test and operational failure rates, the key measures for this project. We find in all cases where data exists for two or more of these columns that the failure rate decreases. A few of the entries in Table 4-2 are described in a little more detail in the following paragraphs for illustration of the data calculations.

The Data Sources 8 and 12 are examples of airborne applications. Failure data for testing was reported on the Advanced Fighter Technology Integration (AFTI) F-16 Program [MACK83] (data source 8). The failure rate represents 15 incidents during the flight test program which involved approximately 180 flight hours. No record of failures observed during the ground operation or ground operating time is available. Most of the failures related to synchronization provisions between the triple redundant computers installed in the aircraft. Software changes were used to correct the problems. It is not clear whether the cause of the failures was due to software deficiencies or to system deficiencies that were overcome by program changes. Thus, the failure rate may be overestimated.

The fault density for data source 12 is derived from two flight control programs, consisting of approximately 40,000 lines of AED code each [HECH83]. The individual fault densities are 0.0018 and 0.0086 respectively.

Data Sources 5, 9, and 14 are examples of strategic applications. The fault density for a real-time C3I system (data source 5) is shown in the table and is the overall fault density (.0085) of four subsystems, with individual measures of 0.004, 0.01, 0.01, and 0.02. The operational software failure rate is the six-month average for the command and control computer associated with the large surveillance radar system.

One real-time operating system application is represented as data source 9 in Table 4-2. It was tested over a 25 month period for an average of 200 hours per month (Figure 4-3), and a total of 270 failures were logged during that interval, equating to the .054 failure rate shown. During the last two months of test (400 hours), eight failures were observed (.02 failure rate). The real-time operating system is part of a classified military software project.

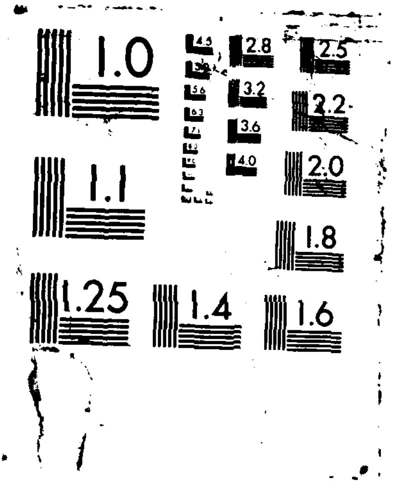
The data for the electronic switching system software (data source 14) pertains to No. 4 ESS as reported in [DAVI81]. An average of 1.6 service-affecting incidents were reported per installation-month during the first quarter of 1980, and 25% of these were attributed to software (an additional 13% were unresolved). The entry in Table 4-2 assumes that there were 0.5 software failures during a 720 hour interval (the system operates 24 hours per day), which includes an allowance for the unresolved incidents. The program involves over 2 million object words, but little is known about other characteristics. The electronic switching systems designed by Bell Laboratories are recognized as representing unusually high hardware and software reliability, and hence it is not surprising that this system has the lowest operational failure rate.

The entries associated with data source 19 under the Tactical Application category are four Real-time C^2 Systems, each involving approximately 20,000 HOL instructions that involved display management and command execution (Projects 1-4 in [MUSA79]). In computing the fault density for these systems which were described in [MUSA79] in lines of object code, it has been assumed that two object instructions are equivalent to one HOL statement. This expansion ratio was used due to the language and computer used for these systems. These four projects were carried out within a single organization and hence it is not too surprising to find a fairly narrow spread of the reliability indicators. The failure rate at the end of test shows a very small range. This characteristic can be controlled in effect by the developing organization (by holding up the release until an acceptably low failure rate is reached).

Data Sets 17 and 21 are the two examples of the Process Control

AD-A190 010 METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION VOLUME 2/3
1(U) SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO
CA J MCCALL ET AL. NOV 87 RADC-TR-87-171-VOL-1
UNCLASSIFIED F30602-83-C-0118 F/G 12/5 NL

6


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

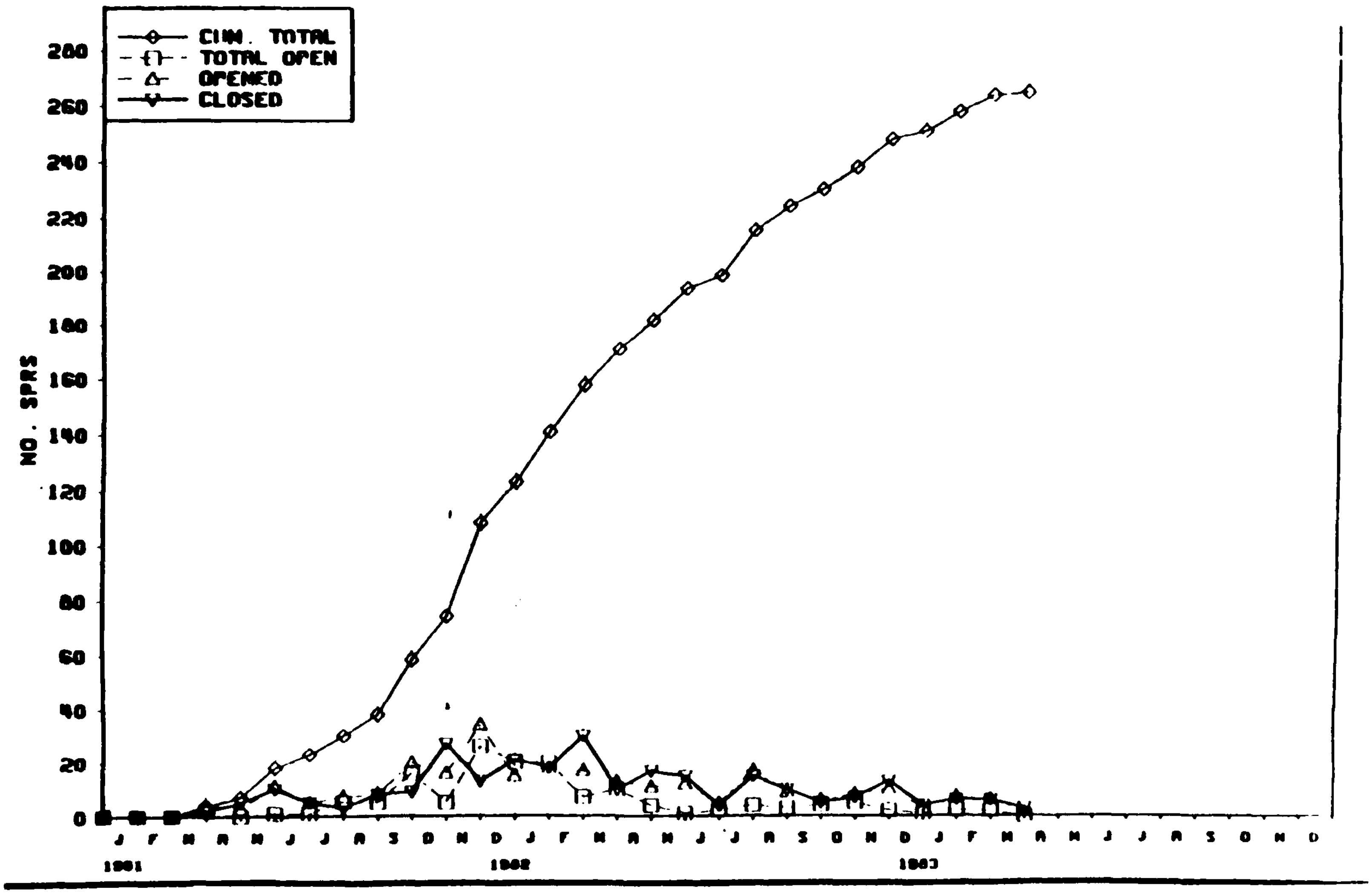


FIGURE 4-3. SPR TEST HISTORY FOR DATA SOURCE 9

Application Category. Data Set 17 is an emergency response information system for a power plant. The fault density represents the number of problems found in the 19,000 lines of code developed for that system. Data Set 21 is an image processing system of over 120,000 lines of code.

Data Sources 6, 7, 13, 15, and 20 are examples of the Production Application category. The in-house interactive program in data source 6 supports a major fiscal agency of the U.S. Government. The data were taken during the last half of 1981 when software outages totaled 3,219 minutes. From related reports, the average software outage lasted 10 minutes, and thus it was assumed that 322 failures occurred. The .total operating time during this period was approximately 3,000 hours.

The fault density and test failure rates for a scientific batch program from the Launch Support Data Base (LSDB) program at Vandenberg AFB [HECH77] is in data source 7. The failure rates were originally provided in execution-time seconds which have an expansion factor of approximately 10 to wall-clock seconds. When this factor is applied and the seconds converted to hours, the failure rates amount to 68 per hour (average) and nine (9) per hour (end of test). This is very much higher than any other data recorded in these columns. Possible reasons for this discrepancy are:

- The early date of these programs (coding took place in 1974 and 1975).
- The test period included unit test which is usually run outside of configuration management and hence excluded from most reported data. This affects primarily the average test failure rate.
- The testing reported here was followed by an acceptance test, the results of which are not included in the data. The end of test failure rate for the acceptance test can be expected to be lower.

The failure rates for data source 13 are derived from System 5, System 17, System 27, and System 40 in [MUSA79]. All of these programs are display oriented and implement math-intensive functions. The fault densities range from 0.0013 to 0.0025. The failure rates range from 0.0044 to 0.13. One of three systems involves over 2 million object instructions, but no other software characteristics are described.

Data Set 15 contains the operational failure rate of a scientific system based on a four month observation of the Viking telemetry data reduction program at the Jet Propulsion Laboratory [MAXW78].

The data for an interactive operating systems (data source 20) were derived from two large computer installations at Standford

University, SLAC and CIT during 1978 - 1980 [IYER81]. There is very little year-to-year variability, and failure rates for the two installations are also quite close (0.024 for SLAC and 0.017 for CIT in 1980). Only unique (new) problems were counted as failures.

One of the Developmental Category data sets (data source 18) is derived from the F-11D data reduction program reported in [WAGO73]. These failure rates were also collected in CPU-seconds and have much higher values in wall-clock hours. This is an even older program than LSDB, and this may help to account for the high failure rate. The program was developed in-house for a data reduction task that was initially assumed to be of very limited scope and then expanded. As is typical under those circumstances, there are very few formal requirements, and the extent of test is largely left up to developer. Thus, a higher failure rate must be expected for support programs under these conditions.

In Section 5 the consolidation of the fault densities and failure rates by application category is presented.

Eventually we hope that enough data may be collected to bypass the use of fault density as a reliability predictor altogether. In that case baseline failure rates achieved (typical) on actual applications would be used. The subsequent prediction metrics would modify this baseline failure rate up or down much like they are intended to do for fault density. The other benefits of collecting the failure rate data shown in the Table 4-2 are:

- The failure rates can be used to track observed results during a development effort. Reliability growth can be tracked according to typical experiences. Lack of progress can be reported to management for their action.

- The empirical relationship between fault density and failure rate can be derived (see Section 5).

4.3.2 Development Environment

This metric is concerned with the effects of the development process which are manifest in the reliability of the software product. Table 4-3 contains a very brief description of the development environments for the projects being used in this study as data sources. Not all development environments are described. For those that are described, they were characterized as an embedded (E), semi-detached (S) or organic (O) environment according to the metric described in Section 3.

4.3.3 Software Characteristics

The software characteristics measurements identified in Section 3 posed a much more significant data collection challenge. To fully satisfy the data collection requirements of many of these measurements, detailed data had to be collected. Examples of data collected for each measurement are provided in the following paragraphs.

The two data sources used primarily for the detailed data collection were the Training System (data source 10) and the Emergency Response Information System (data source 17). These two systems were recently delivered and are being maintained. Key personnel involved in the developments were available for discussions and information if necessary. Documentation and source code were available. The following paragraphs indicate the available sources of data for each of these two systems and a brief description of the system.

DATA SOURCE 10

This system is a large complex tactical training system. The system involved instrumented military exercises where the units participating in the exercise utilized instrumented laser weapons and key players and weapon systems carry transponders so that their location and movement can be tracked via a communications network by computer. Additionally video and communication data is captured. All this data is sent in real time to a computer complex where observers are sitting at workstations observing the exercise. These workstations have graphics displays where the exercise is shown on a terrain map background generated from the Defence Mapping Agency digital terrain data base. The software system that accepts this data, displays it at observer workstations, allows the observers to control displays and stores the data from the complete exercise to facilitate playback for the purposes of debriefing the participants is the data source. The major subsystems of this system are the system software, the display subsystem and the computational component subsystem. The system is a distributed system in that portions of the software run on four VAX 11/780's and 38 workstations with LSI 11/23 processors.

The primary documentation utilized to collect data was:

- Requirements Design Specification - Vol I.
- Requirements Design Specification - Vol II, Part A
- Requirements Design Specification - Vol II, Part B

These documents represented a statement of the requirements, preliminary design and detailed design of the system. Additionally, test documentation, user documentation, and test result documentation were available.

Software Discrepancy Reports were reported throughout the formal testing and operation of the system. Several major enhancements have been made over the last three years. With each enhancement, a formal test and evaluation process was performed. Figure 4-4

TABLE 4-3. DEVELOPMENT ENVIRONMENT, SIZE, AND LANGUAGE CHARACTERISTICS OF THE DATA SOURCES

Data Source	*Application*	*Development Environment Description*	*Size*	*Language*	*Dev Mode*
1	Radar C^2	Build Process Host → Target Cross Compiler Debugging Package Simulator MIL-STD Development Librarian, Source Reformatter Data Set/Used Cross Reference Unit, Integration, Acceptance Testing	136,707	JOVIAL (64%) Assembly (36%)	E
2	Avionics	No Standard 3 Debug Tools Host → Target Cross Compiler Simulator Optimization Tool Module Verification, Integration, System Validation Testing	120,000	Fortran (33%) Assembly (67%)	E
3	Ground Based C^2	NR	115,346	JOVIAL	E
4	C^2	Phased Approach w/Doc not Followed Top Down, Structured Programming Unit, Integration, Acceptance Testing	181,249	Centran	S
	C^2	No Build Approach Formal Testing Through Development, Validation, Acceptance, Integration, and Operational Testing	115,346	JOVIAL	S
5	C^3I	Embedded Development Env.	83,827	HOL	E
6	MIS (Interactive)	NR	NR	NR	O
7	Scientific (Batch)	NR	90,000	Fortran	NR
8	Flight Control	Advanced HW Fault Tolerant Architecture Top Down Design Bottom Up Testing MIL-STD 1679 Like Development	NR	NR	E(1)
9	Real-Time OS	NR	NR	NR	NR
10	Training System	Not MIL-STD Structured Approach Interactive Builds Programmer Workbench	45,702	Fortran	S
11	Mission Planning	NR	4,703	S-Fortran	NR
12	Flight Control	Semi-Detached Development Environment	44,400 43,500	AED AED	S

LEGEND NR – Not Recorded E – Embedded S – Semi Detached O – Organic

TABLE 4-3. DEVELOPMENT ENVIRONMENT, SIZE, AND LANGUAGE CHARACTERISTICS OF THE DATA SOURCES (CONT.)

Data Source	*Application*	*Development Environment Description*	*Size*	*Language*	*Dev Mode*
13	Interactive	NR	2,445,000* 61,900* 128,100* 180,000*	NR NR NR NR *Object	NR
14	Electronic Switching	NR	NR	NR	NR
15	Scientific	NR	NR	NR	NR
16	C^2	MIL-STD Document Batch Card Oriented Compool	183,330	JOVIAL	S
17	RRIS	Commercial Development Modern Tools Extensive Acceptance Test	19,690	Fortran	E
18	Support	NR	NR	NR	NR
19	C^2	NR	21,700* 27,700* 23,400* 33,500*	NR *Object	NR
20	Interactive OS & Batch	NR	NR	NR	NR
21	Image Processing	Phased Approach Top Down Design PDL Standards Used	120,400	Fortran	S
22	Flight Control Mission Preparation	NR	243,883	Fortran (92%) Assembly (8%)	NR
23	Flight Control	NR	46,086 21,022 21,726	Assembly (57%) JOVIAL (39%) Fortran (4%)	NR
24	Support Programs	NR	20,618 38,218	Assembly (94%) Fortran (6%)	NR
25	Satellite C^2	NR	811,630	Fortran	NR
26	Small Business MIS	First Project Involved Structured Development Methodology – Other Three were Informal	10,000 15,000 30,000 30,000	C	O

TABLE 4-3. DEVELOPMENT ENVIRONMENT, SIZE, AND LANGUAGE CHARACTERISTICS OF THE DATA SOURCES (CONT.)

Data Source	Application	Development Environment Description	Size	Language	Dev Mode
27	Ground Based C^2	NR	NR	NR	NR
28	Comm. System	NR	NR	Assembly	O
29	Signal Processing	NR	28,000 36,762	NR NR	S
30	Logistics MIS	NR	1,697,177	Cobol	O
31	Simulation	Incremental Development Modern Design Coding Standards Programming Team	10,038	Fortran	O
32	C^2	Formal Test Approach Formal Development Environment	96,931	JOVIAL	S
33	Simulator	Formal MIL-STD Development Incremental Development	28,564	Fortran (39%) Assembly (61%)	E

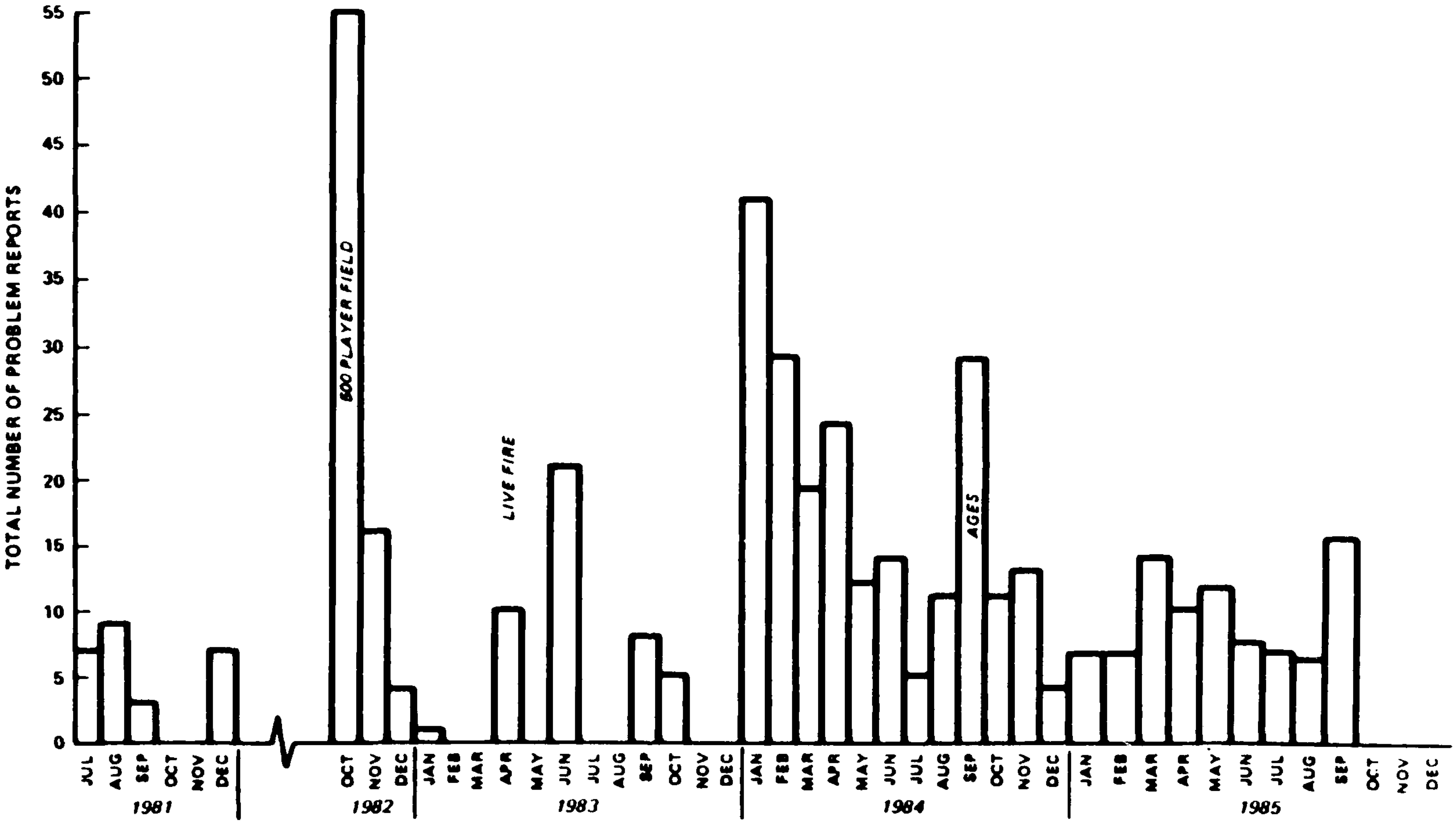


FIGURE 4-4 TRAINING SYSTEM DISCREPANCY

illustrates the discrepancy report frequency over the past five years. The annotated spikes in the frequency correspond to the delivery of new functional capabilities. All discrepancy reports were maintained in a data base. Figure 4-5 illustrates a sample listing from that data base. Test time utilization was recorded during typical periods and this data was used to calculate failure rates experienced.

Source Code was available for collection of code level measurements.

DATA SOURCE 17

This system is an emergency response information system developed to monitor a nuclear power plant. The system monitors various meteorological and radiological information sources, calculates and displays near real-time predictions of atmospheric effluent transport, diffusion and radiological dose estimates, and provided various reports and displays.

The primary document used for data collection was a detailed Technical Specification. This document specified the requirements of the system in significant detail. This project was for a commercial customer and the system was specified to a much greater detail than typical DoD systems.

Discrepancy reports were recorded during formal testing warranty period during which the customer used the system in an operational environment.

Source code was available to collect measurement data also.

4.3.3.1 Anomaly Management (SA), Tracability (ST), and Quality Review Results (SQ) Data Collection

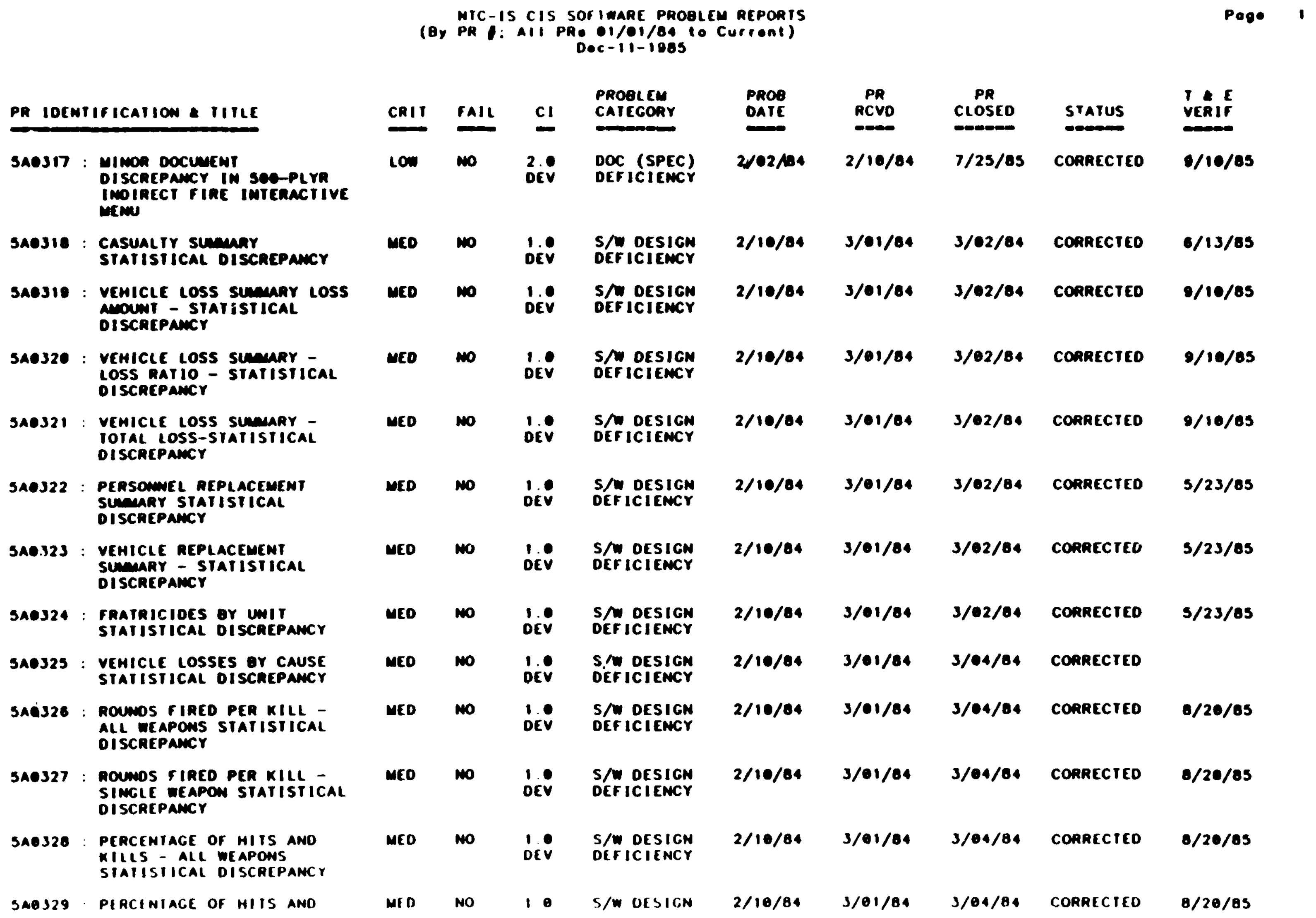
These three measurements required the application of the worksheets contained in RADC TR 85-37. The specific worksheets for the set of three measurements have been incorporated in the Volume II handbook (at Appendicies C and D of Volume II).

4.3.3.2 Language (SL) and Size (SS) Data Collection

These two measurements were more readily available from most data sources. Indications of the languages and sizes for the data sources are in Table 4-3.

4.3.3.3 Extent of Reuse (SU) Data Collection

The SEL data base was used primarily as the data source for this measurement. Nine Programs in that data base have the percentage of reused code indicated. That data is summarized in [HECH83].



```
                    NTC-IS CIS SOFTWARE PROBLEM REPORTS                                          Page   1
                  (By PR #: All PRs 01/01/84 to Current)
                              Dec-11-1985

                                                           PROBLEM          PROB      PR        PR                   T & E
PR IDENTIFICATION & TITLE           CRIT  FAIL   CI        CATEGORY         DATE      RCVD      CLOSED    STATUS      VERIF

5A0317 : MINOR DOCUMENT             LOW   NO     2.0       DOC (SPEC)       2/02/84   2/10/84   7/25/85   CORRECTED   9/10/85
         DISCREPANCY IN 500-PLYR                 DEV       DEFICIENCY
         INDIRECT FIRE INTERACTIVE
         MENU

5A0318 : CASUALTY SUMMARY           MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   6/13/85
         STATISTICAL DISCREPANCY                 DEV       DEFICIENCY

5A0319 : VEHICLE LOSS SUMMARY LOSS  MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   9/10/85
         AMOUNT - STATISTICAL                    DEV       DEFICIENCY
         DISCREPANCY

5A0320 : VEHICLE LOSS SUMMARY -     MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   9/10/85
         LOSS RATIO - STATISTICAL                DEV       DEFICIENCY
         DISCREPANCY

5A0321 : VEHICLE LOSS SUMMARY -     MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   9/10/85
         TOTAL LOSS-STATISTICAL                  DEV       DEFICIENCY
         DISCREPANCY

5A0322 : PERSONNEL REPLACEMENT      MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   5/23/85
         SUMMARY STATISTICAL                     DEV       DEFICIENCY
         DISCREPANCY

5A0323 : VEHICLE REPLACEMENT        MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   5/23/85
         SUMMARY - STATISTICAL                   DEV       DEFICIENCY
         DISCREPANCY

5A0324 : FRATRICIDES BY UNIT        MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/02/84   CORRECTED   5/23/85
         STATISTICAL DISCREPANCY                 DEV       DEFICIENCY

5A0325 : VEHICLE LOSSES BY CAUSE    MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/04/84   CORRECTED
         STATISTICAL DISCREPANCY                 DEV       DEFICIENCY

5A0326 : ROUNDS FIRED PER KILL -    MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/04/84   CORRECTED   8/20/85
         ALL WEAPONS STATISTICAL                 DEV       DEFICIENCY
         DISCREPANCY

5A0327 : ROUNDS FIRED PER KILL -    MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/04/84   CORRECTED   8/20/85
         SINGLE WEAPON STATISTICAL               DEV       DEFICIENCY
         DISCREPANCY

5A0328 : PERCENTAGE OF HITS AND     MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/04/84   CORRECTED   8/20/85
         KILLS - ALL WEAPONS                     DEV       DEFICIENCY
         STATISTICAL DISCREPANCY

5A0329 : PERCENTAGE OF HITS AND     MED   NO     1.0       S/W DESIGN       2/10/84   3/01/84   3/04/84   CORRECTED   8/20/85
```

FIGURE 4-5 SAMPLE BASE LISTING

4.3.3.4 Modularity (M), Complexity (SX), and Standards Review (SR) Data Collection

These measurements required access to the source code or a description of the software at a detailed level. A tool called the Metric Informatin Tracking System (MITS) which is similar in function to the Automated Measurement Tool (AMT) or Automated Measurement Systems (AMS) developed for RADC was used. Figure 4-6 is an example of the output from MITS for elements used to compute the Modularity, Complexity, and Standards Review metrics. Additional source code inspection was required in some cases. Figure 4-7 contains a composite of this data for data source 17. This composite is provided at a CSC level. The number of units contained in each CSC (which was called a process in the system), the number of executable lines of code (for modularity), the number of branches (for McCabe's Complexity), was well as other metric elements for the Standards Review Measurement are shown. The diagonal lines provide separation between the raw metric score (upper left) and the calculated metric element (lower right). Also, indicated is the number of discrepancy reports generated against each CSC.

4.3.4 Test Measurements Data Collection

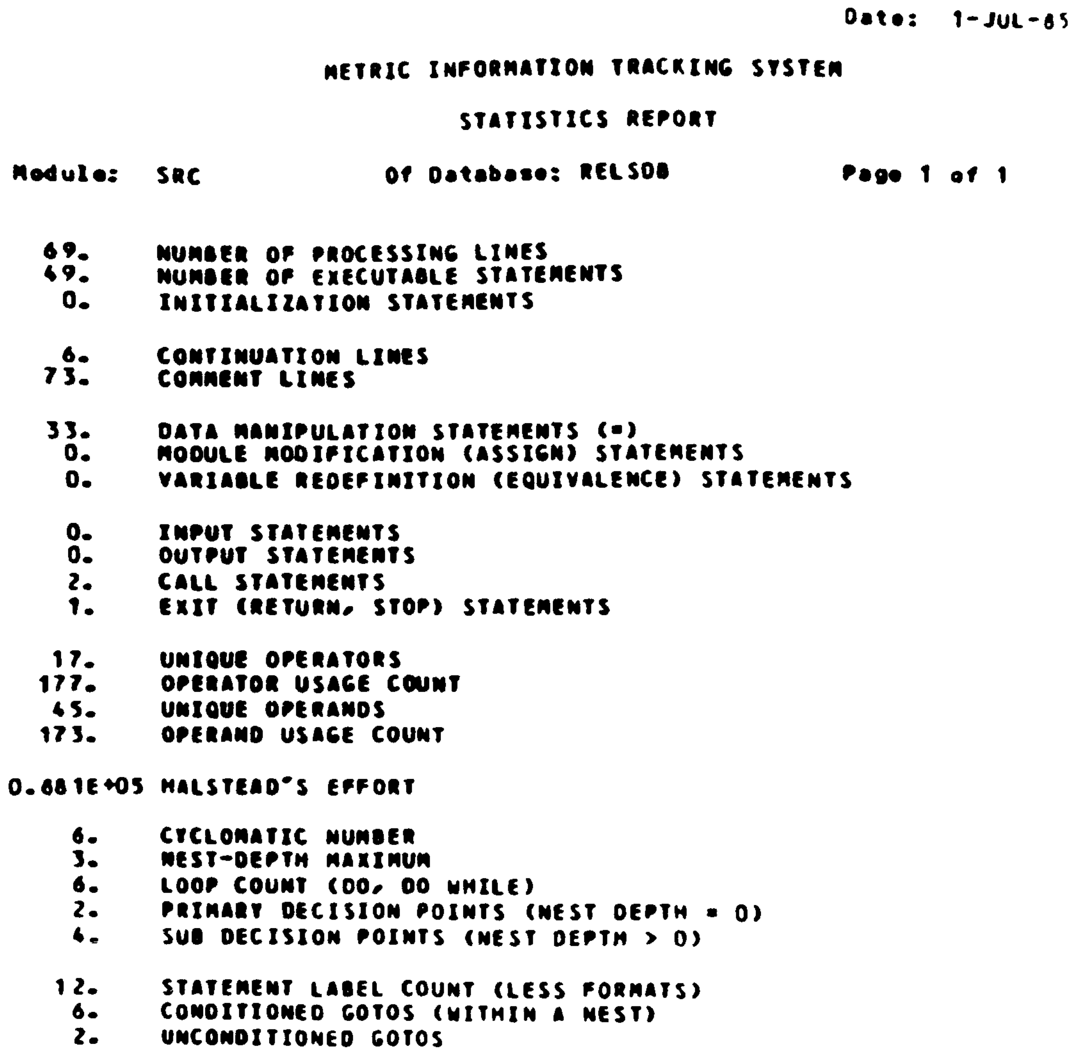
The three test measurements, Test Effort (TE), Test Methodology (TM), and Test Coverage (TC) require different types of data. The Test Effort measurement requires access to labor hour data for the projects and a work breakdown structure accounting system that delineates labor expended testing. The data utilized in this study came from data sources 10 and 17 and represented data collected from project management and test and evaluation management personnel via interviews.

The Test Methodology measurement requires application of RADC TR 84-53. The handbook (Volume II) of that report contained a methodology which when applied recommends testing techniques and tools for particular applications or test objectives. The methodology was applied to data sources 10 and 17. Table 4-4 presents the results of the application of the methodology (path 1).

Test Coverage data was not collected on either of the two detailed data source projects.

4.3.5 Operational Environment Estimation Measurements Data Collection

The two metrics which are used to describe the influence of the operational environment on the reliability estimation, Workload (EW) and Input variability (EV), were also not collected on either of the two detailed data source projects. Data was availablle from [IYER83].



```
                                                          Date:  1-JUL-85

                         METRIC INFORMATION TRACKING SYSTEM

                                 STATISTICS REPORT

Module:   SRC               Of Database: RELSDB             Page 1 of 1


    69.     NUMBER OF PROCESSING LINES
    49.     NUMBER OF EXECUTABLE STATEMENTS
     0.     INITIALIZATION STATEMENTS

     6.     CONTINUATION LINES
    73.     COMMENT LINES

    33.     DATA MANIPULATION STATEMENTS (=)
     0.     MODULE MODIFICATION (ASSIGN) STATEMENTS
     0.     VARIABLE REDEFINITION (EQUIVALENCE) STATEMENTS

     0.     INPUT STATEMENTS
     0.     OUTPUT STATEMENTS
     2.     CALL STATEMENTS
     1.     EXIT (RETURN, STOP) STATEMENTS

    17.     UNIQUE OPERATORS
   177.     OPERATOR USAGE COUNT
    45.     UNIQUE OPERANDS
   173.     OPERAND USAGE COUNT

0.681E+05 HALSTEAD'S EFFORT

     6.     CYCLOMATIC NUMBER
     3.     NEST-DEPTH MAXIMUM
     6.     LOOP COUNT (DO, DO WHILE)
     2.     PRIMARY DECISION POINTS (NEST DEPTH = 0)
     4.     SUB DECISION POINTS (NEST DEPTH > 0)

    12.     STATEMENT LABEL COUNT (LESS FORMATS)
     6.     CONDITIONED GOTOS (WITHIN A NEST)
     2.     UNCONDITIONED GOTOS
```

FIGURE 4-6 SAMPLE METRICS INFORMATION TRACKING SYSTEM OUTPUT

Process (SC)	No of Appl Units	ELOC	No ELOC · 100 MO1 3	No Exits SI1 5A	No Loops SI4.3A	Statement Labels SI4.3A	SI4 6C	Nest Depth SI4.7A	SI4 7B	No. Branches	SI3.1C	SI4 8C	Data Manip. SI4.9C	Data Items SI4.11B	SI4 11C	No. PRs	PRs / ELOC
401 STRTUP	7	68	7 1	0	1	69	1 .142	6	6.2 .885	5	6.16 .88	6 375 .910	52	344	120.381 17 187	0	0
402 QKLOOK	45	1,368	44 .978	36	72	33	38 406 .853	94	28.087 .824	84	24.19 .537	41.225 .916	638	1,023	1 1 .024	1	000731
403 COMMEX	61	1,999	60 .984	59	106	137	55.36 .907	137	33.806 .554	114	40.763 .668	58.271 .955	842	1,495	2 419 .039	8	004002
404 SYEXEC	11	190	11 1	9	5	27	9.109 .828	12	9.0 .818	26	6 096 .554	9 155 .832	61	149	.39 .036	0	0
405 RELEAS	13	672	11 .846	11	59	134	9.432 .725	16	10 33 .794	109	3 838 .295	9 203 .707	386	387	1.081 .084	0	0
406 DCEXEC	112	7,704	87 .777	108	147	234	104.703 .934	271	57 968 .517	133	93.223 .832	110.276 .984	2,436	3,787	44 036 .393	22	002856
407 TPHAND	33	1,141	32 .970	31	16	40	30.476 .923	79	19.178 .581	49	16 877 .511	30.172 .914	318	700	6 194 .188	1	000876
408 TOTPUF	45	1,343	44 .978	38	47	172	38 615 .858	73	31.3 .695	143	22 94 509	40 747 .905	675	1,120	4.526 101	2	001489
409 RRRGSM	113	5,205	100 885	105	249	543	99 115 877	215	73 07 .646	336	49 444 437	103 293 914	2,026	2,453	40 734 360	7	001345
TOTAL	440	19 690	396 9	397	702	1,389	386 216 877	903	268 939 611	999	263 531 598	408 717 928	7,413	11 458	28 943 066	41	002

Total of 25 subroutines not analyzed. Process ICHAND not analyzed

FIGURE 4-7 DATA SOURCE 17 CODE CHARACTERISTICS

TABLE 4-4 APPLICATION OF RADC TR 84-53

	DATA SOURCE 10	DATA SOURCE 17
PATH 1		
STEP 1	TEST CONFIDENCE LEVEL	TEST CONFIDENCE LEVEL
COST	1	0
CRITICALITY	2	3
SCHEDULE	2	1
COMPLEXITY	2	2
DEV. FORMALITY	1	1
S/W CAT.	1	3
ERROR DET.	1	2
TEST COMP.	1	2
	11÷8=1.375 (1)	14÷8=1.75 (2)
STEP 2	SOFTWARE CATEGORY SELECTION SENSOR + SIGNAL PROCESSING (10) / DATA PRESENTATION (14)	SAME
STEP 3 CANDIDATE TECHNIQUE SELECTION	* CODE REVIEWS * ERROR DETECTION STRUCTURE ANALYSIS * PROGRAM QUALITY ANALYSIS PATH ANALYSIS * DOMAIN TESTING DYNAMIC PATH ANALYSIS * PERFORMANCE MEASUREMENT * REAL TIME TESTING	* CODE REVIEWS * ERROR DETECTION STRUCTURE ANALYSIS * PROGRAM QUALITY ANALYSIS PATH ANALYSIS * DOMAIN TESTING DYNAMIC PATH ANALYSIS * PERFORMANCE MEASUREMENT * REAL TIME TESTING PARTICIPATION ANALYSIS DATA FLOW GUIDED TESTING ASSERTION CHECKING RANDOM TESTING MUTATION TESTING
SELECT TOOLS	TEST RESULT ANALYZER * TEST DOC. WRITER * TEST MANAGEMENT SYSTEM * TEST DRIVER AUTOMATED VERIFICATION SYSTEM * PERFORMANCE MONITOR	TEST RESULT ANALYZER * TEST DOC. WRITER * TEST MANAGEMENT SYSTEM * TEST DRIVER AUTOMATED VERIFICATION SYS. * PERFORMANCE MONITOR ASSERTION CHECKER DATA FLOW ANALYZER RANDOM TEST GENERATOR MUTATION ANALYSIS SYSTEM
RATIO OF TECHNIQUES/TOOLS USED TO RECOMMENDED:	$\frac{10}{15}$	$\frac{11}{24}$

* TECHNIQUES OR TOOLS USED ON PROJECT

4.3.6 Test and Operational Test Time Data Collection

Data was collected to facilitate calculation of failure rate. Table 4-5 provides data from data source 10 identifying CPU hours spent testing and corresponding discrepancy reports recorded. Data Source 17 did not have this type of data recorded, however the system has been running for over a year at the customer site 24 hours a day and only 41 software discrepancy reports have been reported over that time period.

4.4 DATA COLLECTION LESSONS LEARNED

As in all data collection activities, lessons were learned that would have enhanced the efficiency with which the data collection was performed and the quality of the data collected. Some of the specific lessons learned during this effort were:

- In a research effort such as this, there is a tendency to want to continue to refine the metrics and identify new ones - even after data collection activities have proceeded. At some point in any project, even a research effort, a data definition document should be developed which specifically identifies the data elements to be collected. This document should be driven by the data collection objectives or goals and each data element identified should be related to a specific objective. In a research effort, other elements, not specifically related to an objective, can be identified for collection in support of future analyses that might change a metric or create a new one.

- A companion document to the data definition document should also be prepared. This document should be a data collection guide. This guide should at a minimum:

  - Identify the sources for data collection.

  - Provide all forms and reports for data collectors.

  - Identify any data base management systems to be used for storage of the data collected.

  - Provide a case study or example to illustrate data collection approach.

  In addition, the guide might provide any implementation specifics for this project, for example:

  - Programming language-specific examples, and

  - Documentation-specific examples.

TABLE 4-5 MONTHLY TOTALS FOR DISCREPANCY REPORTS AND TEST CONTROL SHEETS

MONTH	YEAR	TOTAL # PRs	ELAPSED TEST TIME (HOURS)
JUNE	1981	0	01.00
JULY		7	08.32
AUGUST		9	02.25
SEPTEMBER		3	13.12
OCTOBER		0	46.08
NOVEMBER		0	67.50
DECEMBER		7	04.67
OCTOBER	1982	55	NR
NOVEMBER		16	NR
DECEMBER		4	NR
JANUARY	1983	1	NR
APRIL		10	NR
JUNE		21	NR
SEPTEMBER		8	NR
OCTOBER		5	NR
NOVEMBER		0	17.18
JANUARY	1984	41	39.23
FEBRUARY		20	NR
MARCH		12	12.37
APRIL		18	61.07
MAY		12	24.42
JUNE		14	34.90
JULY		5	NR
AUGUST		11	NR
SEPTEMBER		29	20.08
OCTOBER		11	15.33
NOVEMBER		13	28.87
		332	

LEGEND: NR= NOT RECORDED

- Retrieval of data from existing data bases such as the DACS or SEL data bases are usually more time consuming than anticipated. The data available is usually not as well organized, cross-referenced, or defined as well as expected. Therefore, this data should be depended upon only as support data or complementary data, to support analyses of more detailed data collected.

- All data collected should be stored in a centralized, controlled data base. The data should be placed in electronic format to facilitate later analyses and retrieval. This format should be compatible with the DACS.

It is recommended that future data collection activities include these above specific requirements.

5.0 DEMONSTRATION AND VALIDATION OF SOFTWARE RELIABILITY MEASURES

5.1 APPROACH TO THE ANALYSIS OF THE CANDIDATE SOFTWARE RELIABILITY PREDICTION AND ESTIMATION MEASURES

The overall approach taken to analyzing the data collected is shown in Figure 5-1. Each measurement was individually analyzed to determine its relationship to the reliability numbers calculated for the various data sources. An attempt was made in most cases to hold as many other variables constant while analyzing the apparent relationship one measurement had.

The objectives of our analyses were to:

- Determine or establish the relationship each measurement has with the reliability numbers.

- Demonstrate that relationship via the data sources available during this project.

- Statistically validate the relationship if the data sample is sufficient.

- Document additional data collection requirements, metrics or analyses that should be done.

In investigating the relationships, as many past studies that were appropriate were used. Simple straightforward relationships were investigated first prior to more complicated relationships. Thus in some cases, recognizing that the use of the measurement was to provide a sample or first cut reliability prediction (e.g., Application Type which is identified via a table look up), the simple average and variance of the fault density experienced with each application category was calculated. In other cases, linear regression analysis was used to statistically determine the relationship of the metric to the reliability numbers. In a few cases, non-linear regression analysis was used.

5.2 ANALYSIS OF THE DATA

The analyses performed are described in the following paragraphs. The analyses are presented organized by measurement. Results and findings for each metric are presented in these paragraphs. Overall results are described in paragraph 5.3.

5.2.1 Application Type (A)

All of the data sources were used in analyzing the Application Type. The goals of this analysis were to establish baselines and provide an initial reliability prediction number. This initial

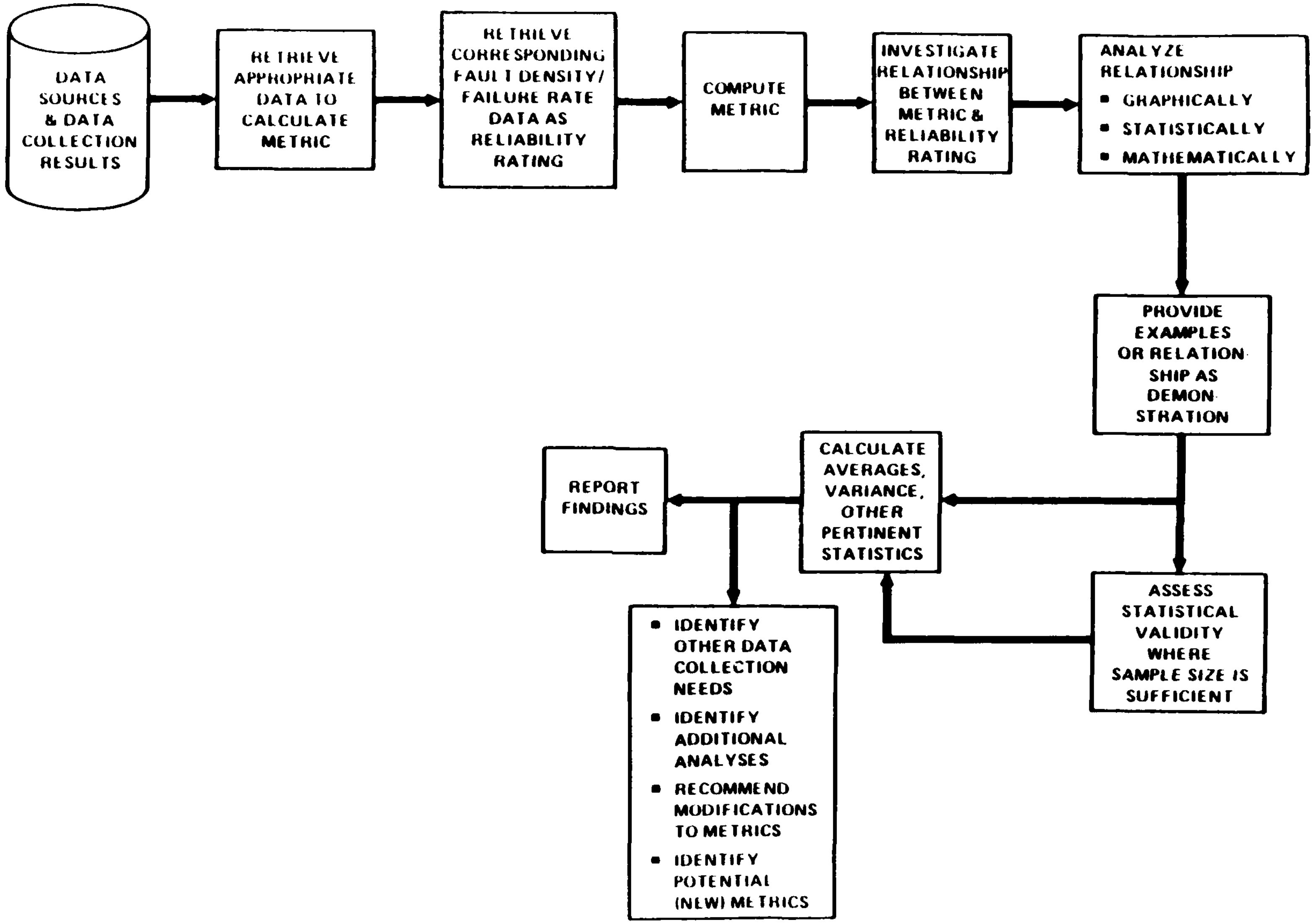

DATA SOURCES & DATA COLLECTION RESULTS
RETRIEVE APPROPRIATE DATA TO CALCULATE METRIC
RETRIEVE CORRESPONDING FAULT DENSITY/ FAILURE RATE DATA AS RELIABILITY RATING
COMPUTE METRIC
INVESTIGATE RELATIONSHIP BETWEEN METRIC & RELIABILITY RATING
ANALYZE RELATIONSHIP
■ GRAPHICALLY
■ STATISTICALLY
■ MATHEMATICALLY
PROVIDE EXAMPLES OR RELATION-SHIP AS DEMON-STRATION
ASSESS STATISTICAL VALIDITY WHERE SAMPLE SIZE IS SUFFICIENT
CALCULATE AVERAGES, VARIANCE, OTHER PERTINENT STATISTICS
REPORT FINDINGS
■ IDENTIFY OTHER DATA COLLECTION NEEDS
■ IDENTIFY ADDITIONAL ANALYSES
■ RECOMMEND MODIFICATIONS TO METRICS
■ IDENTIFY POTENTIAL (NEW) METRICS

FIGURE 5-1. THE APPROACH TO THE ANALYSES

prediction number could be viewed as an industry average or baseline for the particular application. Table 5-1 provides averages for each sample by Application Type. This table is a summarization of Table 4-2. Indicated in the table is the number of systems for which data was collected for that Application Type. The total number of systems in the data base was 59. Of these 59, the number of source lines of code were reported for 49 amounting to over 5 million lines of code. The average fault density indicated is a weighted average, i.e. it is the total number of errors found divided by the total number of lines of code for all systems in that application category. The fault density by system indicated is an average of the fault densities reported for each system, i.e. the system size is not taken into account. A standard deviation for the average fault density by system is given in parentheses. The failure rates shown are the average failure rate during formal testing, the failure rate at the end of the test period and operations failure rate. The failure rate is in units of failure per computer operation hour.

The airborne applications consisted of eight different data sources (systems). One large system written primarily in assembly language in the early 1970s (data source 2 - [FISH79]) had a fault density reported of .017. Two others written in AED (both approximately 40,000 lines of code each) were real-time closed-loop flight control systems and reported fault densities of .0086 and .0018 [HECH83]. Four others were flight control programs on-board the ALCM or B-1B [HECH83] and had fault densities reported as .0029, .011, .021, and .027. A last system, the digital flight control system on the Advanced Fighter Technology Integration (AFTI) F-16 program, reported a .08 failure rate (.08 failures per operational flight hour) during flight testing.

The strategic systems data consists of 25 different systems. Most of these systems are military C3I systems, ground-based C^2 systems, NASA ground stations, or communication switching systems. The range in fault densities reported was .054 to .0001 and in failure rates, .C28 to .0007. The later failure rate (.0007) was the most reliable system reported in the data base (data source 14). Many of the systems in this application category were of significant size, over 100,000 lines of code.

The tactical systems data consists of 5 systems. These ranged from four command and control applications (data source 19) to a tactical training system (data source 10). The four C^2 projects each involved between 10,000 to 20,000 HOL instructions performing display management and command execution in a command and control system (Projects 1 - 4 in [MUSA79]). Individual data for these projects are presented in Table 4-2. The fault density entry in Table 5-1 is an average of these four plus the other tactical system. These four projects were carried out within a single organization and hence it is not too surprising to find a fairly narrow spread of the reliability indicators. The training system (data source 10) was described in Section 4. It has all

TABLE 5-1 APPLICATION TYPE AVERAGES FOR FAULT DENSITY AND FAILURE RATE

APPLICATION	TOTAL NO OF SYSTEMS	FAULT DENSITY			FAILURE RATE		
		TOTAL LOC (NO. OF DATA POINTS)	AVERAGE FD	FD BY SYSTEM (STD DEV)	AVG TEST (NO. OF DATA POINTS)	END TEST (NO. OF DATA POINTS)	OPNS (NO OF DATA POINTS)
AIRBORNE	8	540.617 (7)	.009	.013 (.009)	.08 (1)	--	--
STRATEGIC	25	1,793,831 (21)	.009	.0092 (.014)	.34 (4)	.0427 (5)	.0108 (5)
TACTICAL	5	88,252 (5)	.005	.0078 (.0061)	2.6 (5)	.64 (5)	.108 (5)
PROCESS CONTROL	2	140,090 (2)	.0017	.0018 (.0003)	--	--	.007 (1)
PRODUCTION	14	2,575,427 (12)	.0027	.0085 (.0095)	68 (1)	1.85 (5)	.198 (4)
DEVELOPMENTAL	5	97,435 (4)	.011	.012 (.009)	170 (1)	21 (1)	--
TOTAL/AVERAGE	59	5,235,652 (49)	.0058	.0094 (.011)	21 (12)	2.1 (16)	.1 (14)

of the ingredients of an operational tactical system. Its reported fault density was .0016. Failure rate data was also captured for this system. It was a 1.04 average during testing, .63 at end of test and .18 during operation.

The Process Control Application Type was only represented by two data sources. This Application Type was created to distinguish between the critical nature of the airborne, strategic and tactical applications and the production center and developmental applications. It represents some aspects of each of the above two groups. The two systems used were an Emergency Response Information System (data source 17), described in Section 4, and an Image Processing System (data source 21). Fault densities reported for these two were .002 and .0016 respectively.

The Production Systems category was represented by fourteen data sources. These ranged from an interactive operating system at a university (data source 20) to interactive commercial and military systems (data source 13) to an in-house system running financial management systems (data source 6) to a Launch Support Data Base program at Vandenberg AFB (data source 7) and telemetry processing for the Viking Project at JPL (data source 15). These systems ranged in size between 10,000 lines of code to one system that was 1,697,177 lines of code. About half of these systems were interactive, transaction processing type systems while the other half were simply batch processing systems.

The Developmental Systems are represented by five systems. One is data source 18 which is a data reduction system and two are the support programs described in [HECH83] (data source 24). The two other systems (data sources 31 and 33) are simulators. The failure rates reported on data source 18 were very high (170 for test average and 21 for end of test). This is the only failure rate data reported for this category, so the average may be biased high.

Table 5-1 illustrates the improvement in reliability expected from failure rate average test, end of test, and operational.
The data collected exhibits, on the average, a ratio of approximately 9 to 1 between the average failure rate during test to the failure rate observed at the end of test and a ratio of approximately 7 to 1 between the failure rate at the end of test and the operational failure rate (see Table 5-2). The averages are calculated from Table 4-2 for these data sources where failure rates are reported for each of these pairwise comparisons. The range in the ratios of average failure rate during test to end of test failure rate is 1.7:1 to 41.2:1. If the one system that exhibited the 41.2:1 ratio is eliminated then the average ratio is 5:1 with a range between 1.7:1 and 8.9:1. The range in the ratios of end of test failure rate to operational failure rate 2.5:1 to 11:1 with the calculated average of 7:1. These ratios are potentially valuable estimation parameters to allow rule of thumb estimates of failure rates to be expected at end of test or during operation based on the

TABLE 5-2. TRENDS IN FAILURE RATES

APPLICATION CATEGORY	AVERAGE TEST: END TEST	END TEST: OPERATIONS
STRATEGIC	12.6 : 1	2.5 : 1
TACTICAL	4.1 : 1	5.9 : 1
PRODUCTION	37 : 1	9.3 : 1
DEVELOPMENTAL	8 : 1	--
AVERAGE	8.6 : 1 (5 : 1) *	7.3 : 1

* Average after eliminating one extreme ratio

observed average failure rate during testing. Data is needed for the Airborne and Process Control Categories to complete this table.

Another relationship which we had hoped to observe was specific differences in either fault density or failure rate exhibited by the Application Categories. In Table 5-1 it can be seen that the Airborne and Strategic Application categories exhibited the same average fault density (.009), the developmental category exhibited the highest average fault densities (.011), the process control category exhibited the lowest average fault density (.0017), and the production system and tactical categories exhibited fault densities of .0036 and .0027 respectively. Additional data sources in the process control category needed to confirm it as having the lowest fault density. Our expectations that the highly critical systems (exhibited by airborne, strategic, and to some degree tactical systems) would exhibit lower fault densities than other categories were not met. Where our expectations were consistent with the findings was in observed failure rates. The strategic system category had an average failure rate of .0108 during operation. The airborne category only had failure rate data available from one data source and it was an average during test. It was .08 which was significantly lower than the .34 average test failure rate exhibited by the strategic systems. Thus we could expect a better operational failure rate for the airborne systems. The tactical system operational failure rate (.108) was next in the expected hierarchy of failure rates. The production systems category with a failure rate .198 was next with the developmental systems (a failure rate of 21) last using the end of test failure rate reported for one data source. These differences are further illustrated if failure rates are calculated for each data source in Table 4-2 for which failure rates for end of test or operations were reported. Using these, averages for each application category are shown in Table 5-3. In this table, the categorization scheme recommended by Hecht is also shown based on the processing time constraints of the systems. Using this scheme, clear differences in the failure rates observed are exhibited. The real time applications had an average failure rate of .0048, the on-line (interactive, transaction processing) applications had an average of .016, the batch process applications had an average of .02 and the one developmental support application had an average of 21. This categorization scheme seems most promising.

Figures 5-2a, b, c, and d presents the data in Tables 5-1, 5-2, and 5-3 graphically. Two general phenomena are observed. One is that the reliability of the more time critical systems is higher than less time critical systems (Figure 5-2c). This same concept potentially holds for the more functionally critical systems having the higher reliability (Figure 5-2b) but more data is required.

The other phenomenon is the reliability growth illustrated

TABLE 5-3 FAILURE RATE BY APPLICATION CATEGORY

APPLICATION TYPE	TIME CONSTRAINT CATEGORIES				
	REAL TIME	ON-LINE	BATCH	SUPPORT	AVERAGE
AIRBORNE	.08 (1)				.08
STRATEGIC	.019 (10)				.019
TACTICAL	.108 (5)				.108
PROCESS CONTROL	.007 (1)		--		.007
PRODUCTION		.016 (6)	.02 (3)		.017
DEVELOPMENTAL				21 (1)	21
AVERAGE	.0048 (17)	.016 (6)	.02 (3)	21 (1)	

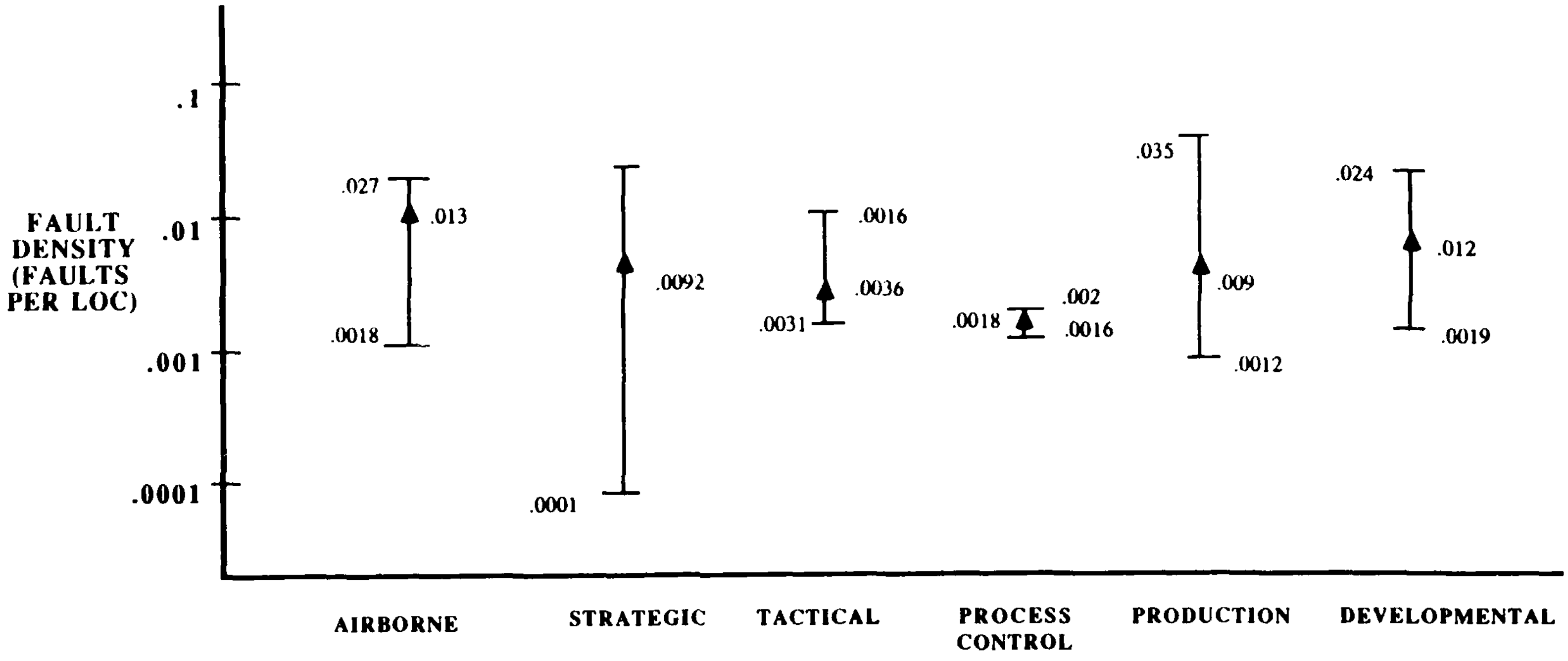


FIGURE 5-2A FAULT DENSITY BY APPLICATION

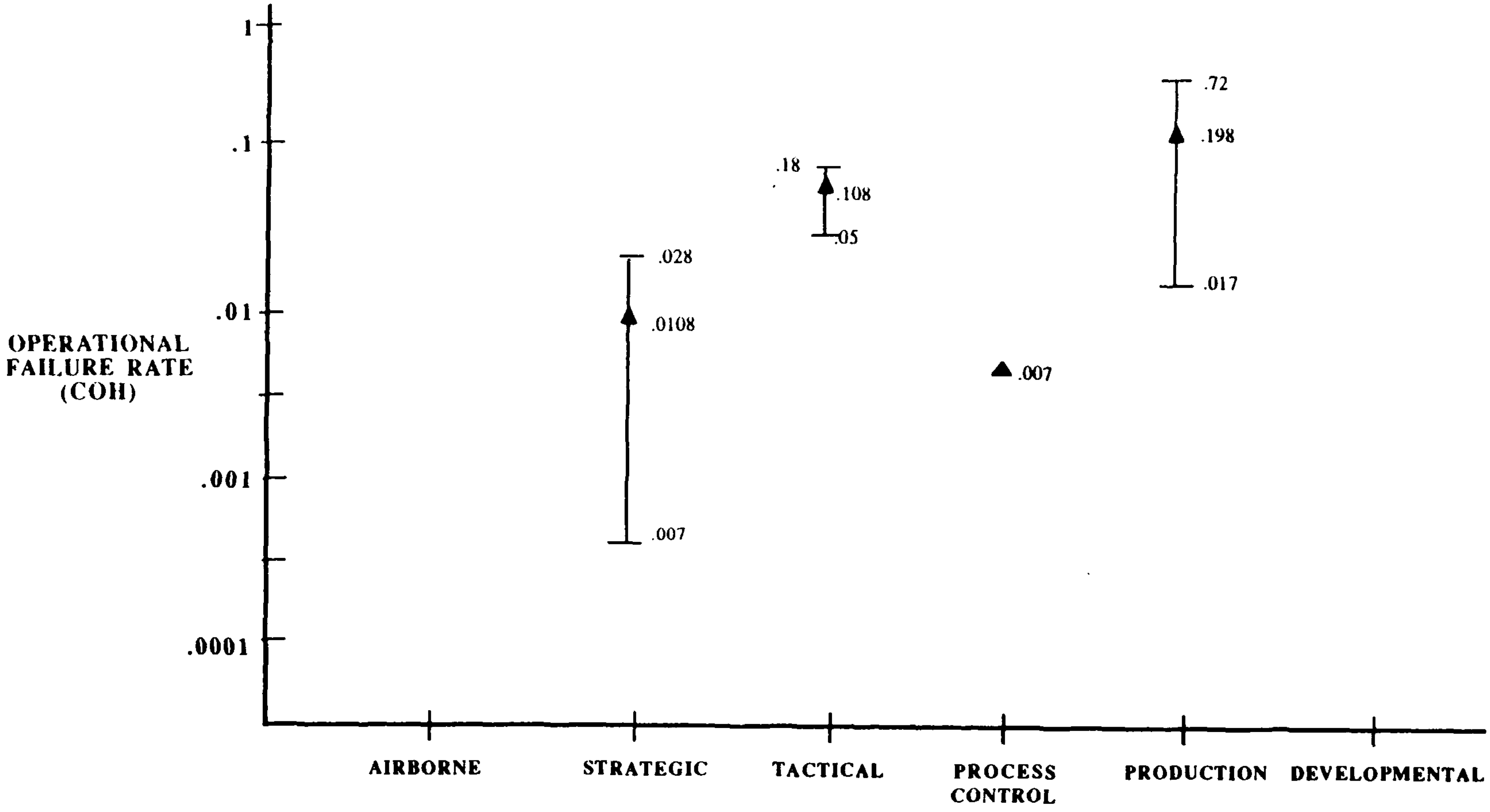


FIGURE 5-2B OPERATIONAL FAILURE RATE BY CATEGORY

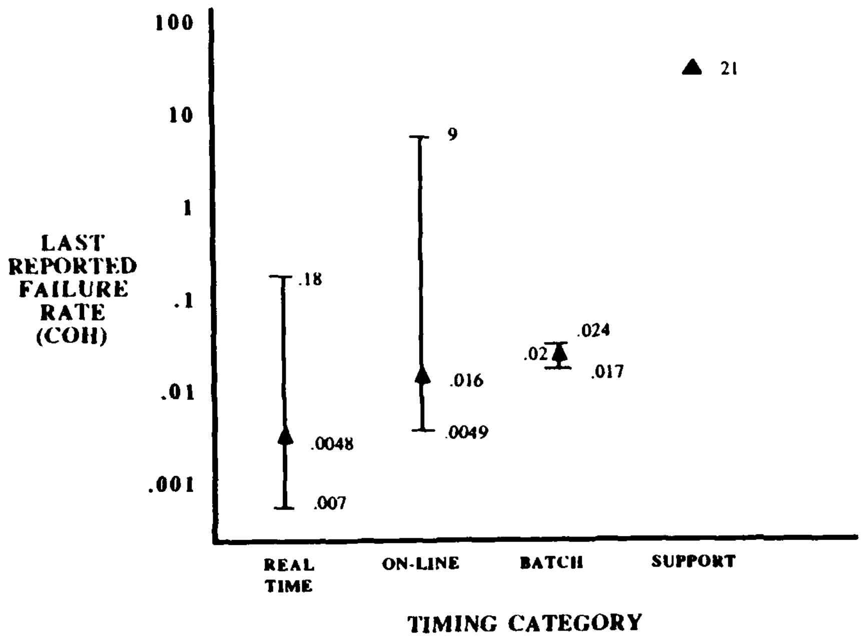


FIGURE 5-2C
FAILURE RATE BY
TIMING CATEGORY

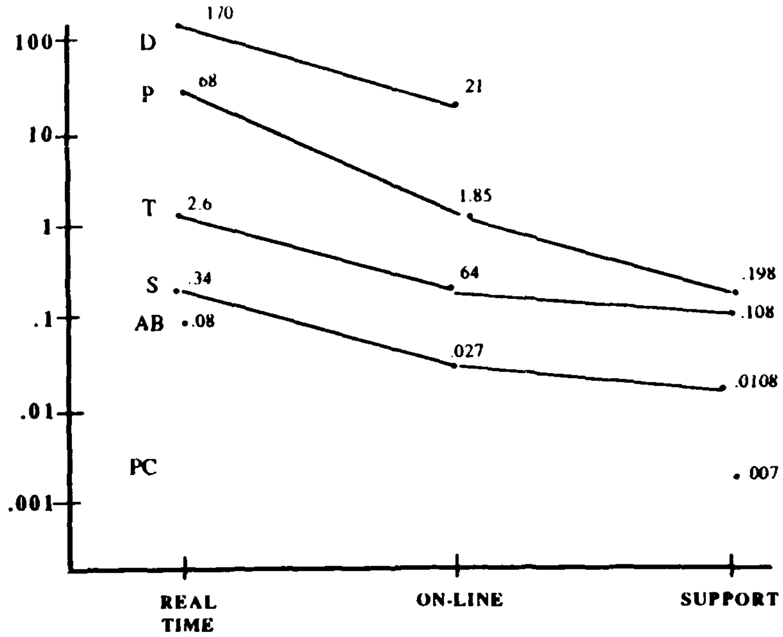


FIGURE 5-2D
FAILURE RATE DURING PHASES
BY APPLICATION

through the test phase into operations (Figure 5-2d). All failure rates are in Computer Operation Hour (COH).

An expected relationship not illustrated by the data was related to fault density and application type (Figure 5-4a). It appears that the more critical systems which are developed typically with more formality still exhibit approximately the same fault densities as the non-critical systems. This probably happens because they are subjected to more formal testing. The differences show up once the system is fielded when the critical systems exhibit the lower failure rate since most of their faults have been removed. The non-critical systems still contain many faults and have higher failure rates.

The basic purpose of these analyses was to develop an initial set of baselines, which are in Table 5-1 and Table 5-2.

5.2.2 Development Environment (D)

As previously discussed, the development environment as well as the software implementation are viewed as contributors to the fault density and are evaluated primarily against that measure. To establish the prediction factors for the development environment, two approaches are available:

- Gross statistics -- determine the fault density of many software projects in each class; and

- Selective comparison -- determine the fault density of comparable projects in each class.

Figure 5-3 illustrates the data available from the data sources relating the Development Mode metric to fault density. Note within each category of Development Mode there is a scale. This scale represents the rating derived from the checklist described in Section 3 (Table 3-7). That checklist identifies what techniques and tools were employed during the development. The rating is derived from a ratio of the items checked divided by the total numbers of items, ie. if 19 items are checked of the total 30 the rating is .5. From the limited data available, there appears to be a relationship which is intuitively supported; the more formal tools and techniques employed, the more faults found during the development phase. The relationships exhibited by the data in Figure 5-3 are:

FD = .109d - .04 for Embedded
FD = -.008d + .009 for Semi-detached
FD = -.018d - .003 for Organic

where d is the rating of the development approach using the checklist (Table 3-7).

These relationships represent taking a gross statistical technique. To have confidence in these relationships, data from

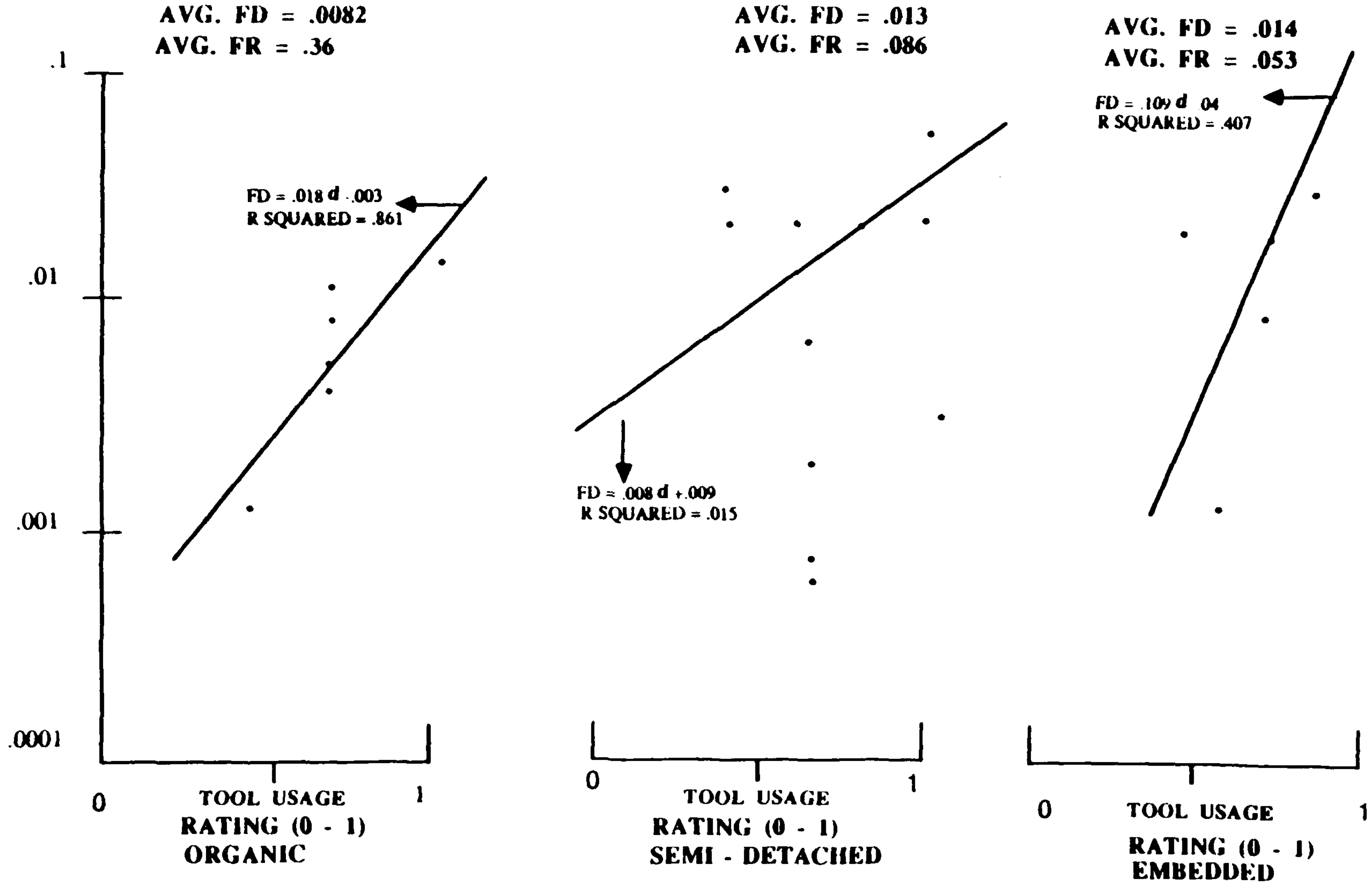


FIGURE 5-3 DEVELOPMENT MODES RELATIONSHIP TO FAULT DENSITY

a significant number of projects (approximately 30 in each category) would have to be gathered. The current correlations are not statistically significant but do exhibit an intuitive relationship. Figures 5-4, 5-5, and 5-6 illustrate the relationships.

Selective comparisons were also made to assess if more insight could be provided of the affect of the development mode on software reliability.

One such comparable observation will be used as an example. An organic environment is represented by the real-time flight control program listed as data source 12 in Table 4-2. The flight control software represented by this data was produced by a group within the flight control equipment manufacturer's organization having a considerable familiarity with the application. The real-time command and control software represented by data source 5 in Table 4-2, in comparison, was produced in an embedded environment. Both software products involved approximately 40,000 lines of code, run under tight timing constraints, and incorporate modern programming practices.

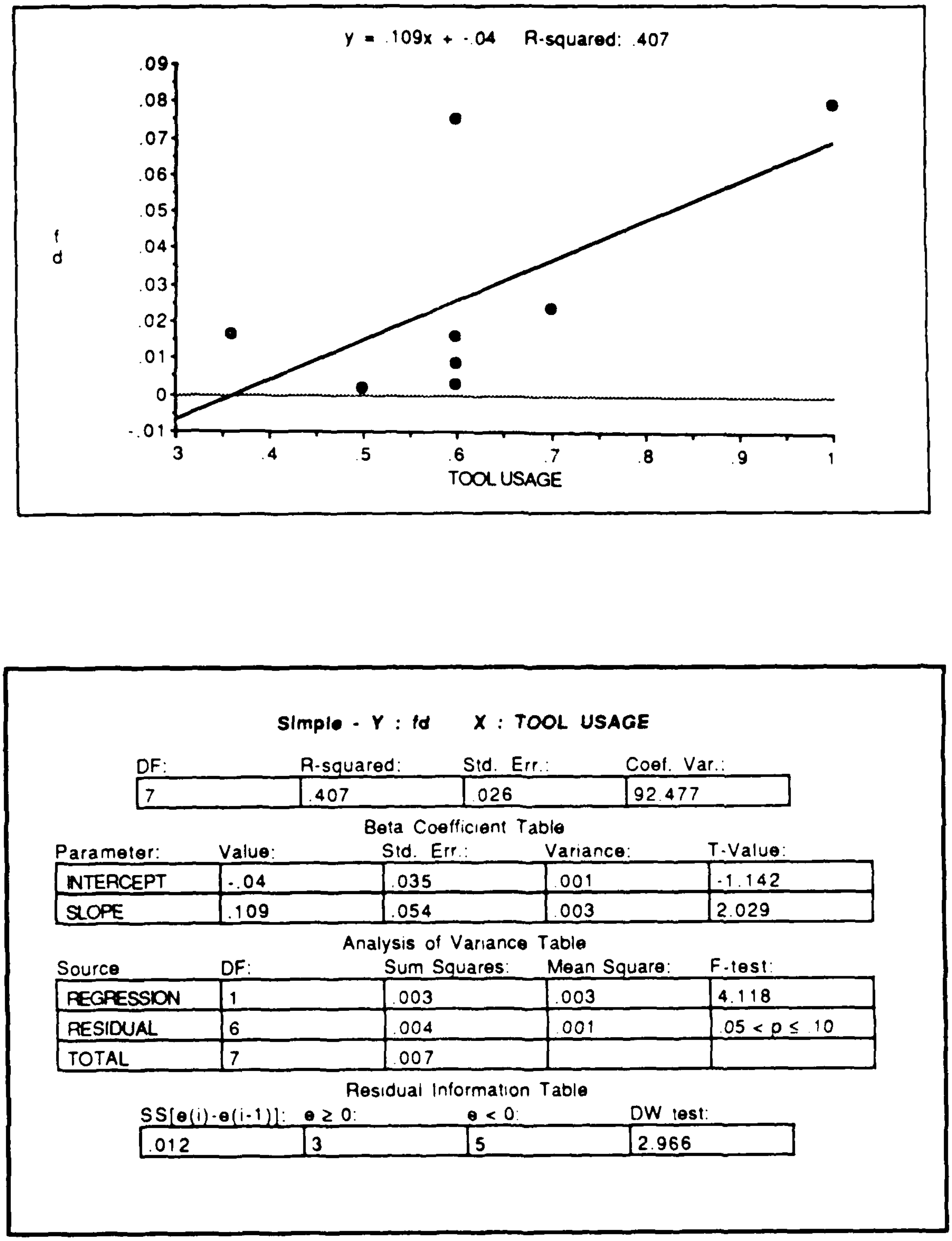
The fault densities for these two examples are:

- Organic environment -- 0.005
- Embedded environment -- 0.0085

If the observations reported here carry through for a larger sample, the embedded environment will then be assigned a fault density multiplier that is 0.0085/0.005 = 1.7 greater than that of the organic environment. Since it is desired to have the unity value of the parameter for a neutral environment, the organic development environment will be assigned a value of 0.76 and the embedded environment a value of 1.3, the ratio of these being 1.7. As a check, the average fault density for the embedded data sources used in Figure 5-3 is .014 and for the organic data sources .0082 which is consistent with the 1.7 ratio (.014/.0082) calculated above. These summary relationships between the development modes will be used to establish a basic multiplier for the development environment metric. This multiplier will be modified if information is available to complete the checklist. In this case, the equation presented earlier are used.

5.2.3 Software Characteristics

Each of the metrics described in Section 4 were analyzed against the fault density data collected. Some of these metrics were analyzed at the system or subsystem level, others at the CSC or unit level. Where the analyses were performed at the CSC or unit level, data sources 10 and 17 were used.



Simple - Y : fd X : TOOL USAGE

DF:	R-squared:	Std. Err.:	Coef. Var.:
7	.407	.026	92.477

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-.04	.035	.001	-1.142
SLOPE	.109	.054	.003	2.029

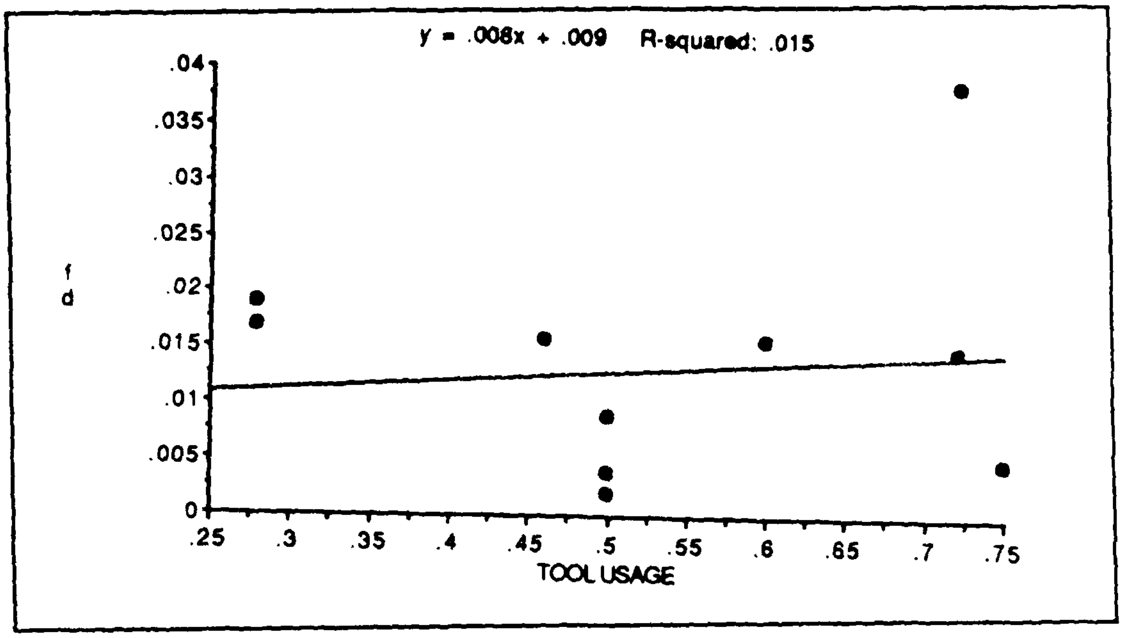
Analysis of Variance Table

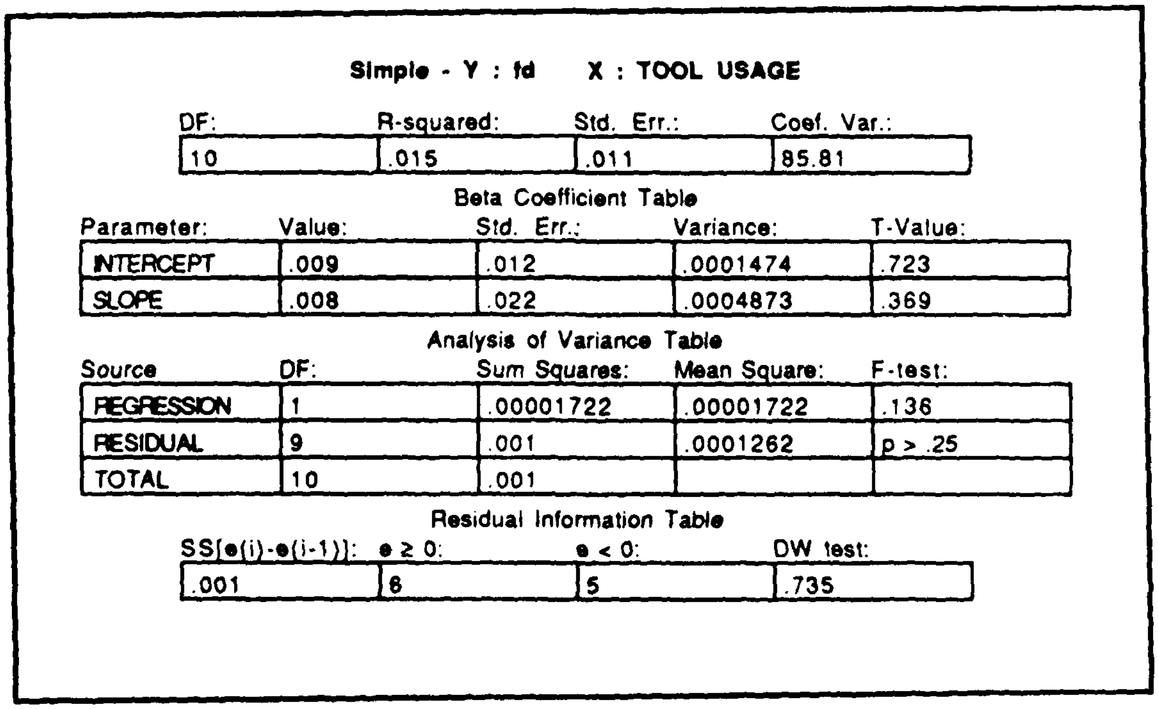
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.003	.003	4.118
RESIDUAL	6	.004	.001	.05 < p ≤ .10
TOTAL	7	.007		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
.012	3	5	2.966

FIGURE 5-4 EMBEDDED MODE ANALYSIS





Simple - Y : fd X : TOOL USAGE

DF:	R-squared:	Std. Err.:	Coef. Var.:
10	.015	.011	85.81

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	.009	.012	.0001474	.723
SLOPE	.008	.022	.0004873	.369

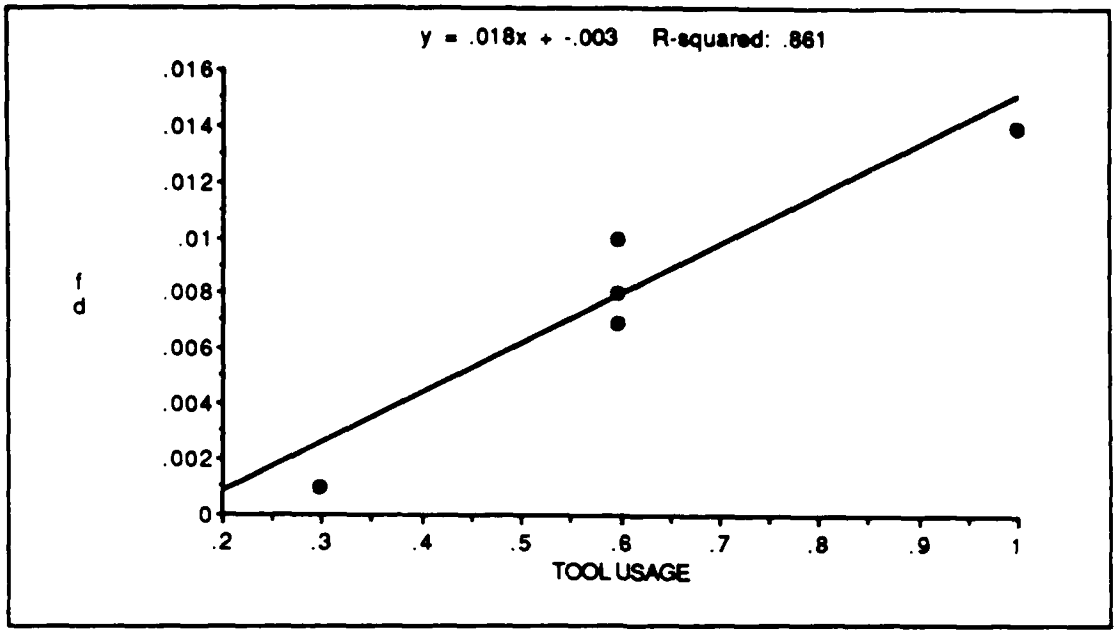
Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.00001722	.00001722	.136
RESIDUAL	9	.001	.0001262	p > .25
TOTAL	10	.001		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
.001	6	5	.735

FIGURE 5-5 SEMI-DETACHED MODE ANALYSIS



Simple - Y : fd X : TOOL USAGE

DF:	R-squared:	Std. Err.:	Coef. Var.:
5	.861	.002	21.628

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-.003	.002	.000005516	-1.175
SLOPE	.018	.004	.00001308	4.973

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.00008034	.00008034	24.733
RESIDUAL	4	.00001299	.000003248	.005 < p ≤ .01
TOTAL	5	.00009333		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
.00002316	2	4	1.782

FIGURE 5-6 ORGANIC MODE ANALYSIS

5.2.3.1 Anomaly Management, Traceability and Quality Review

The Anomaly Management metric and Quality Review metric scores as applied to data source 10 are in Table 5-4. These metrics were applied at a CSC (process) level since the design documentation was written with that orientation. The results of the statistical analysis of these scores versus the fault density recorded are in Figures 5-7 and 5-8. As can be seen, neither analysis provided significant results, i.e., results that could be used for prediction. Both metrics demonstrated a correlation with fault density, i.e. as the metric score went up, the fault density went down, but the relationship was not significant statistically. The Quality Review results were disappointing. The results expected should have supported Lipow's findings in [LIPO79] where units which had many design problems also were ones that had the most implementation problems.

Further investigation revealed the following:

- Processes with an AM score greater than .6 had a fault density of .0008.
- Processes with an AM score between .4 and .6 had a fault density of .001.
- Processes with an AM score less than .4 had a fault density of .004.

This analysis lends itself to developing a metric with a multiplier based on the above findings. A conservative approach will be taken assigning a multiplier of .9 for an AM score greater than .6, 1 for an AM score between .4 and .6, and 1.1 for a score less than .4. A similar relationship was found with the Quality Review metric. Utilizing a QR score .5 as a divider, QR scores higher had an average fault density of .0007 and QR scores lower had an average fault density of .0016. Again utilizing a conservative approach, a multiplier of 1.1 was assigned to SQ if the metric score was lower than .5.

An attempt was made to assess traceability. Without the use of a formal requirements specification language such as PSL/PSA or SREM or a significant expenditure of labor to establish a traceability matrix utilizing a tool such as RTT, this was very difficult to do within the scope of this project for systems as large as data source 10 and 17.

Additional analyses are needed to establish whether these metrics can be used as predictors. See Section 7 for recommendations and plans.

5.2.3.2 Software Implementation Characteristics

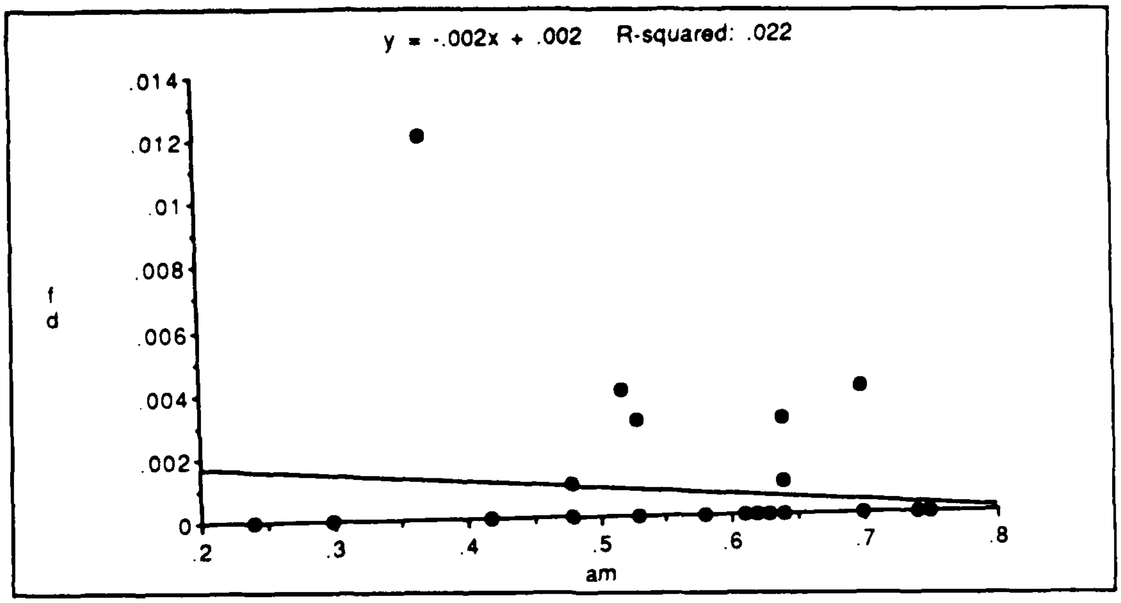
Table 5-5 contains a summarization of the data collected from data sources 10 and 17 to analyze the software implementation

TABLE 5-4 ANOMALY MGMT AND QUALITY REVIEW METRIC VALUES FOR DATA SOURCE 10

PROCESS	ANOMALY MGMT	QUALITY REVIEW	#PR	FD
101	.63	.43	0	0
102	.24	.32	0	0
103	.52	.40	9	.0044
104	.70	.86	5	.0044
105	.37	.36	8	.012
106	.70	.87	0	0
107*	--	--	--	--
108*	--	--	--	--
109	.53	.81	9	.0028
110	.53	.39	0	0
111	.53	.81	0	0
112	.70	.44	0	0
113	.53	.42	0	0
114	.70	.47	0	0
115*	--	--	--	--
116*	--	--	--	--
117*	--	--	--	--
118*	--	--	--	--
119**	.48	.38	0	0
120**	.62	.44	0	0
121**	.30	.30	0	0
122**	.30	.30	0	0
201	.58	.43	0	0
202	.70	.86	0	0
203	.64	.48	0	0
204	.53	.30	0	0
205	.58	.43	0	0
206	.63	.47	0	0
207	.64	.89	3	.0012
208	.48	.44	0	0
209	.61	.87	3	.001
210	.48	.84	25	.0031
211	.64	.88	0	0
212**	.64	.89	1	.0012
301*	--	--	--	--
302	.63	.88	0	0
303	.64	.88	0	0
304	.64	.89	2	.001
305	.64	.89	2	.0014
306	.64	.69	0	0
307	.48	.43	0	0
308**	.74	.94	0	0
309**	.75	.94	0	0
310**	.42	.78	0	0
311**	.42	.76	0	0
312**	.42	.94	0	0
313**	.75	.94	0	0
314**	.42	.93	0	0

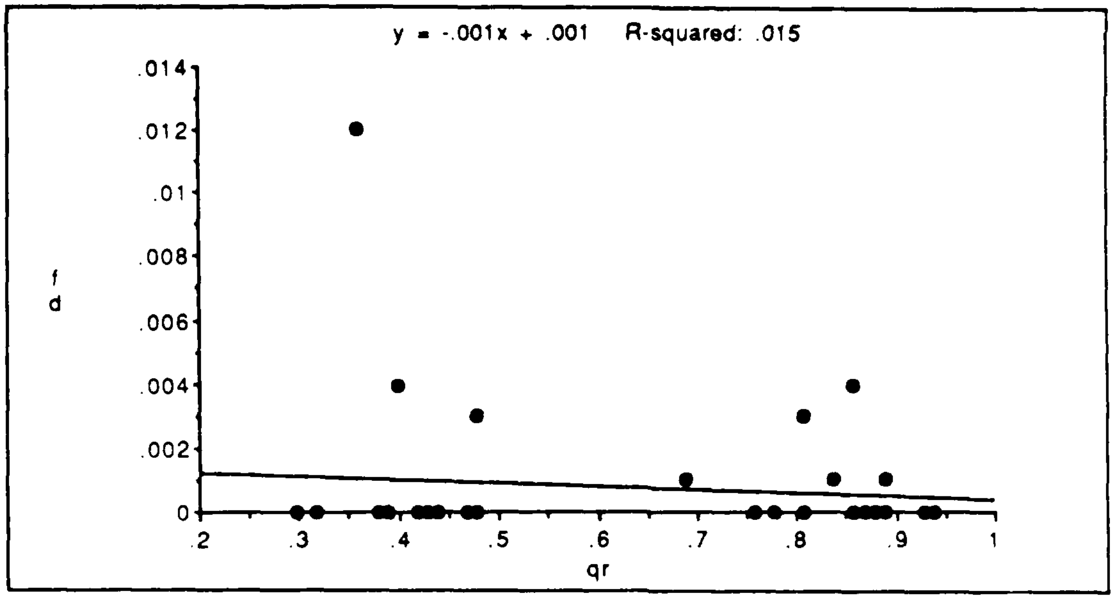
* Process not available at design

**Process either deleted or combined with other processes in implementation



```
                     Simple - Y : fd     X : am

           DF:            R-squared:      Std. Err.:     Coef. Var.:
          | 40           | .022          | .002          | 277.443      |

                            Beta Coefficient Table
Parameter:     Value:          Std. Err.:      Variance:       T-Value:
| INTERCEPT   | .002          | .001          | .000002232    | 1.429       |
| SLOPE       | -.002         | .003          | .000006682    | -.946       |

                           Analysis of Variance Table
Source         DF:             Sum Squares:    Mean Square:    F-test:
| REGRESSION  | 1             | .000003941    | .000003941    | .896        |
| RESIDUAL    | 39            | .0001716      | .000004401    | p > .25     |
| TOTAL       | 40            | .0001756      |               |             |

                          Residual Information Table
           SS[e(i)-e(i-1)]:  e ≥ 0:        e < 0:          DW test:
          | .0002441     | 10            | 31            | 1.423        |
```

FIGURE 5-7 ANOMALY MANAGEMENT STATISTICAL ANALYSIS



Simple - Y : fd X : qr

DF:	R-squared:	Std. Err.:	Coef. Var.:
40	.015	.002	278.56

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	.001	.001	8.948E-7	1.511
SLOPE	-.001	.001	.000001916	-.759

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.000002556	.000002556	.576
RESIDUAL	39	.000173	.000004436	p > .25
TOTAL	40	.0001756		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
.0002498	10	31	1.444

FIGURE 5-8 QUALITY REVIEW STATISTICAL ANALYSIS

TABLE 5-5 SUMMARIZATION OF DATA COLLECTED TO ANALYZE THE SOFTWARE IMPLEMENTATION CHARACTERISTICS

PROCESS	DATA-MANIP	DATA ITEMS	S131C	S146C	S147B	S148C	S1411C	#PR	#PR/ELOC
101	38	87	9.142	7.953	7.700	9.684	.804	0	0
102	30	87	6.250	7.158	6.400	7.531	.958	0	0
103	1129	1352	44.872	53.695	45.340	57.637	4.106	9	.0044
104	1045	4034	59.294	26.345	78.953	77.437	-863.419	5	.0044
105	2353	474	10.950	16.158	11.410	16.685	-4.183	8	.0120
106	1882	1471	38.110	40.600	37.910	44.670	23.654	0	0
107	1166	2036	52.420	56.825	21.654	56.711	5.699	3	.0013
108	6	16	.490	1.670	1.500	1.709	1.287	0	0
109	1235	1970	47.531	63.503	44.200	66.072	9.233	9	.0028
110	36	112	4.571	5.352	5.000	5.714	2.946	0	0
111	154	299	14.090	15.810	13.490	17.357	-.676	0	0
112	61	144	6.010	10.710	9.660	10.177	.878	0	0
113	17	64	7.410	7.323	7.830	8.550	4.253	0	0
114	29	79	1.955	3.284	2.666	3.626	-1.547	0	0
115	696	1130	89.432	92.636	50.086	92.570	37.025	3	.0012
116	34	49	.111	.977	.200	.911	.455	0	0
117	7	28	.200	.888	.500	.851	-.037	0	0
118	6	22	.250	.903	.500	.903	.290	0	0
201	54	216	4.223	10.308	10.030	9.869	-3.796	0	0
202	501	1117	16.434	41.113	36.180	40.854	8.726	0	0
203	304	489	12.300	25.232	20.882	25.444	8.503	0	0
204	30	80	4.130	5.907	5.330	6.401	2.398	0	0
205	102	172	4.730	5.676	5.830	6.396	-.499	0	0
206	12	53	2.767	3.604	4.000	4.547	1.638	0	0
207	1087	2176	87.735	98.164	73.312	102.707	-66.846	3	.0012
208	209	420	15.440	16.995	8.350	17.601	2.804	0	0
209	866	1795	65.817	84.313	79.000	92.465	21.845	3	.0010
210	3046	4735	114.100	182.781	116.507	189.743	21.135	25	.0031
211	107	282	7.469	15.554	14.820	15.452	2.978	0	0
301	1113	2539	86.629	121.980	125.320	127.745	-30.260	0	0
302	572	1268	18.688	44.157	54.000	44.352	-1.517	0	0
303	459	877	11.587	21.773	28.000	21.876	3.321	0	0
304	515	1255	22.973	47.885	61.000	48.329	-6.495	2	.0014
305	876	1631	23.212	44.388	52.330	45.503	12.159	2	.0010
306	276	680	7.501	23.853	31.000	22.829	-6.224	0	0
307	461	936	22.898	53.226	45.310	57.260	17.437	0	0
401	52	344	6.160	1.000	6.200	6.375	-120.381	0	0
402	638	1023	24.190	38.406	28.087	41.225	1.100	.0007	.0007
403	842	1495	40.763	55.360	33.806	58.271	2.419	.0040	.0040
404	61	149	6.096	9.109	9.000	9.155	.390	0	0
405	366	387	3.838	9.432	10.330	9.203	1.091	0	0
406	[illegible]	3787	93.223	104.703	57.968	110.276	44.036	22	.0029
407	700	700	16.877	30.476	19.178	30.172	6.194	1	.0009
408	1120	1120	22.940	38.615	31.300	40.747	-4.526	2	.0015
409	2453	2453	49.444	99.115	73.070	103.293	40.734	7	.0013

TABLE 5-5 SUMMARIZATION OF DATA COLLECTED TO ANALYZE THE SOFTWARE IMPLEMENTATION CHARACTERISTICS (Cont'd)

PROCESS	# SUBR	ELOC	EXITS	LT100-ELOC	LOOPS	LABELS	BRANCHES	NEST-DEPTH
101	10	111	9	10	7	16	6	14
102	8	115	7	8	4	11	5	6
103	60	2030	58	56	65	80	61	79
104	87	1141	3	86	65	565	157	72
105	18	665	17	16	17	34	36	32
106	46	3235	44	38	169	64	47	38
107	57	2385	56	54	190	11	18	209
108	2	47	1	2	2	6	7	3
109	70	3187	68	62	140	125	143	144
110	6	108	4	6	1	13	15	5
111	18	399	17	18	15	26	19	27
112	12	197	11	12	5	25	34	13
113	9	100	7	9	3	11	8	8
114	4	102	3	4	7	7	10	9
115	93	2428	93	92	139	36	52	232
116	1	90	0	1	1	2	8	5
117	1	27	0	1	0	3	4	2
118	1	31	1	1	1	3	3	2
201	13	247	11	13	9	33	40	12
202	48	1490	46	47	23	123	215	61
203	29	1032	25	28	26	65	65	50
204	7	145	6	7	2	14	16	9
205	7	175	6	7	9	22	10	8
206	5	103	4	5	1	11	14	5
207	106	2570	100	105	112	166	139	190
208	18	541	17	18	6	25	12	46
209	102	3106	98	99	78	355	346	137
210	203	7961	196	145	470	515	459	479
211	19	399	18	19	11	51	54	26
301	145	2854	150	140	78	403	409	135
302	54	1470	53	53	28	277	324	17
303	28	1066	24	25	6	235	232	1
304	61	1470	60	60	25	272	343	17
305	54	1953	53	49	35	340	330	19
306	31	744	30	30	13	163	203	12
307	60	1978	57	57	23	116	161	88
401	7	68	0	7	1	69	5	6
402	45	1368	36	44	72	33	84	94
403	61	1999	59	60	106	137	114	137
404	11	190	9	11	5	27	26	12
405	13	672	11	11	59	134	109	16
406	112	7704	108	87	147	234	133	271
407	33	1141	31	32	16	40	49	79
408	45	1343	38	44	47	172	143	73
409	113	5205	105	100	249	543	336	215

characteristics. Data Source 10 CSC's are identified by Processes 101-307. Data Source 17 CSC's are identified by Processes 401-409. The following paragraphs describe our analyses for each metric.

Language

The Language metric was evaluated in [HECH83] for a significant sample of programs. Typical data from that study are shown in Table 5-6. For post-1977 programs, the average fault density of assembly programs was found to be .0103 and that of HOL programs was found to be .0075 (both are here expressed as a ratio of faults to the source statements whereas in the reference they are given as percentages of equivalent assembly statements). If HOL is used as the baseline (metric = 1), assembly language code therefore carries a multiplier of .0103/.0075 = 1.4.

TABLE 5-6.
EFFECT OF LANGUAGE ON RECENT PROGRAMS

Program Attribute	Assembly	HOL
Number of Programs	6	15
Program Size*	100K	1,124k
Average Fault Density**	.0103	.0015
Range of Fault Density	.0015 - .0521	.0001 - .0086

* Equivalent executable assembly statements
** Fault density = No. of faults per line of exectuable code

Most of the High Order Language (HOL) programs included in this sample were written in FORTRAN. Two programs were written in the AED programming language, generally considered to represent a more primitive type of HOL, and these had an average fault density of .0052. Because of the small size of that sample it may be premature to establish a differentiation based on the type of HOL in which the program is implemented. None of the programs in that sample were written in a block-structured HOL. PASCAL and Ada programs should be examined and their reliability attributes examined to determine whether they differ significantly from those of FORTRAN programs.

For earlier programs, the following fault densities in percent are reported in [NELS78]:

FORTRAN (18)	.0151
COBOL (9)	.0129
PL/1 (2)	.0333
CENTRAN (3)	.0194
Assembly (24)	.0266

TABLE 5 7 PRIOR USE OF CODE FOR SELECTED SEL PROGRAMS

SYSTEM/SEL NO.	SIZE	% FORTRAN	%ASSEM	%REUSED	%MOD	FAULT DENSITY
AEM-MANPOWER ALLOC/2	26488	84	16	1.8	9.2	.0042
ISEEB - INT SUN EXPL/5	122718	87	13	14.1	6.3	.0007
PANORAMIC ATT SCAN/6	198965	86	14	6.3	17.9	.0001
SEASAT/10	109147	76	24	28.9	5.6	.0005
FOXPRO/35	16997	72	28	26.8	14.5	.001
DYNAMICS EXPLORERA/36	140812	87	13	18.7	14.4	.0016
DYNAMICS EXPLORER B/37	128444	85	15	19.6	13.0	.0014
DEB DETERMINISTIC/38	13829	79	21	23.8	13.5	.0029
MAGSAT Nr REAL-TIME/49	28900	85	15	26.8	6.2	.0001
TOTALS	786,300	84	16	16.3	12.1	.0009

The number of programs involved is indicated in parentheses after each language. The unweighted average fault density of the four high order languages is .0202; the average weighted by the number of programs involved is .016. The ratio of assembly to HOL fault densities is 1.3 and 1.6, depending on the method of averaging.

Using fifteen more projects from the current data base that were implemented in a single language each, the following additional fault densities are reported:

FORTRAN (6)	.017
JOVIAL (2)	.001
COBOL (1)	.0012
C (4)	.0085
AED (2)	.005
ASSEMBLY (4)	.0148

Again, calculating the average HOL fault density to be .0114 and dividing this into the Assembly language fault density (.0148), a ratio of 1.3 is derived. This is in very good agreement with the findings reported above and indicates that the multiplier for assembly language is reasonably firm.

Reuse

The extent of prior use is documented for many programs in the Goddard-SEL data base. Table 5-7 lists the percentage of re-used and modified lines of code of programs for which the fault density had been computed in [HECH83]. These programs were developed in a reasonably uniform environment between 1977 and 1980. They comprise from 14,000 to 200,000 executable statements. The primary language is FORTRAN with assembly segments that range from 13% to 28% of the code.

Two analysis were conducted on this data sample. The first one considered only the percentage of re-used code and resulted in the following findings (Table 5-8):

TABLE 5-8
PRIOR USE OF CODE FOR SELECTED SEL PROGRAMS

Percent Re-used	No. of Systems	Avg. Fault Density by System	Weighted Avg. FD
< 10	2	0.00215	.00058
10 - 20	3	0.0012	.00125
> 20	4	0.0011	.00068

The second analysis considered re-used code and 50% of the modified code (together termed Re/Mod Code) and yielded the following results (Table 5-9):

TABLE 5-9
REUSED AND MODIFIED CODE IMPACT ON FAULT DENSITY

Percent Re-used	No. of Systems	Avg. Fault Density by System	Weighted Avg. FD
< 15	1	0.0042	.0042
15 - 30	2	0.0003	.0003
> 30	6	0.00125	.0012

Both analyses did not find a conclusive relationship between fault density and re-used code. From the limited data currently available, no predictive relationship could be developed. Other programming environments need to be explored in order to assess if representative and accurate predictor can be developed.

Size of Code

Comparisons of fault density for programs of different size are currently available from three sources, [HECH83], [NELS78], and this study. The former includes 16 programs (at least 75% of each coded in HOL), all of which were developed between 1978 and 1980 in a disciplined programming environment; [NELS78] comprises 52 programs developed prior to 1977 in a variety of languages (including many assembly programs) and programming practices. This study includes most of the systems in [HECH83] plus additional ones. The effect of size on fault density is shown in Table 5-10. The data collected during this study is portrayed graphically in Figure 5-9.

TABLE 5-10.
EFFECT OF SIZE OF CODE

	Fault Density, Percent Source:		
Program Size (DSLOC)	HECH83	NELS78	This Study
< 10K	.001*	.034	.054*
10K - 49.9K	.0036	.0084	.0074
50K - 99.9K	.0021*	.0087**	.0195
> 100K	.001	.0124	.0088

* Class comprises a single program
** Excluding one program at .14.

The overall trend seems to indicate that large programs have a lower fault density than small ones which is counter-intuitive. Possible explanations are a greater amount of re-used code in large programs and a more disciplined programming environment. In the NELS78 data set, it is quite likely that the large programs made more use of HOLs.

Figure 5-9 could be misleading because of the two extremely large systems. Figure 5-10 is a regression using the same data except those two large systems. This figure shows even less correlation and highlights the fact that size does not appear to be related to the fault density.

At a CSC level within a system, the relationship is more consistent with expectations. Figure 5-11 illustrates the correlation found between size and fault density in data source 10 where size of CSCs are plotted.

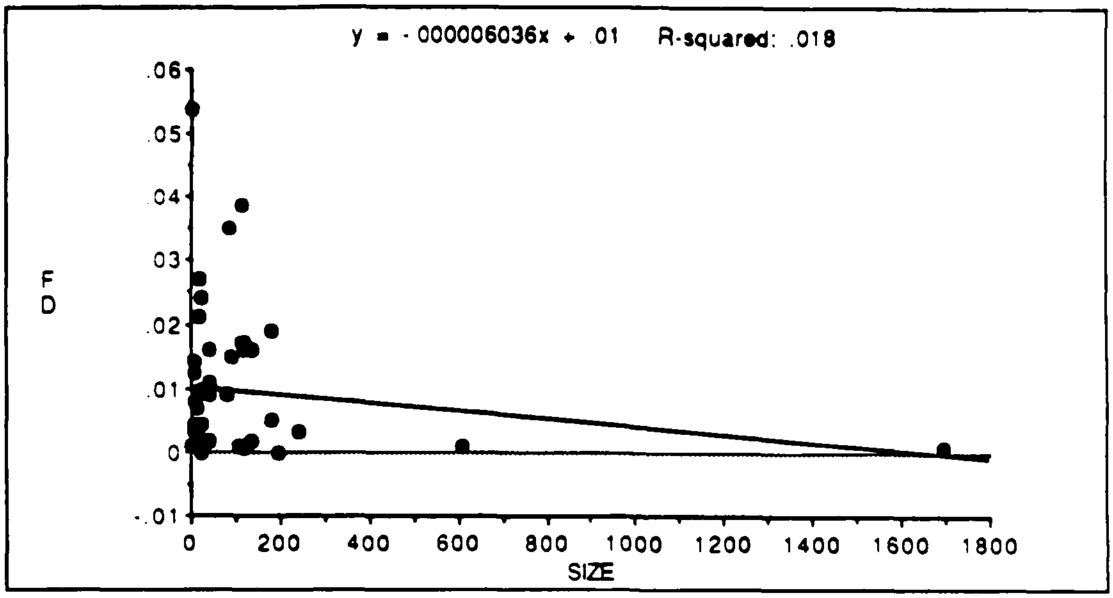
Modularity

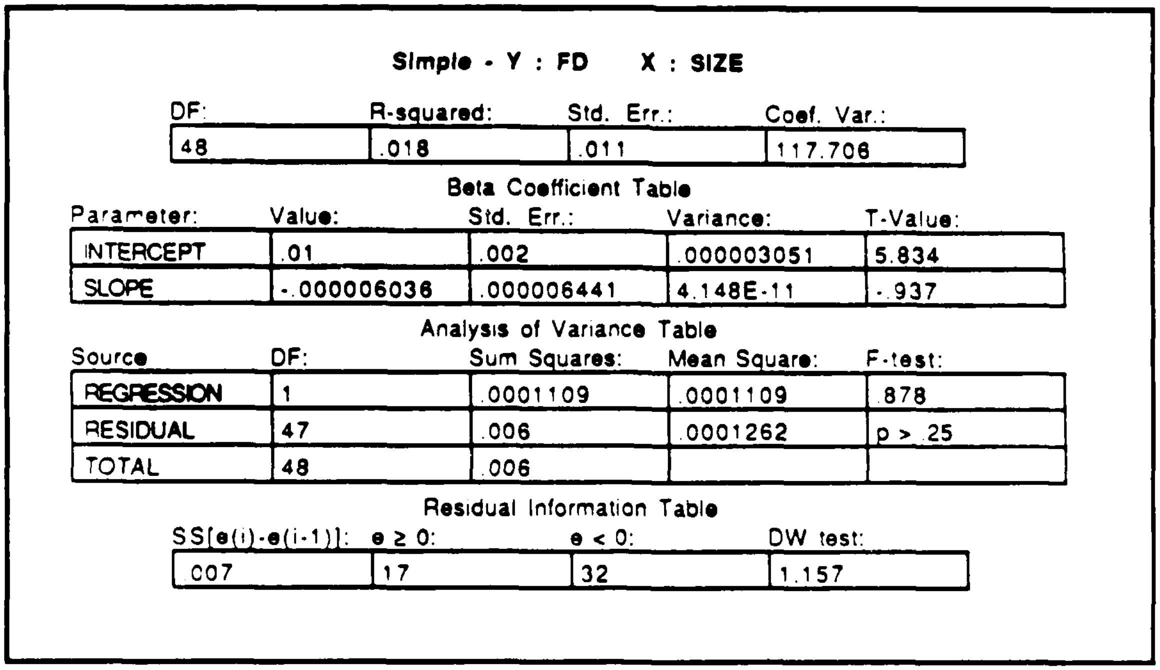
The effect of module size on fault density has been evaluated on the basis of data from data sources 1, 4, 10, 11, 17, 21, and 29. Data source 1 is predominantly written in JOVIAL/J3 and was tested over a three year period that ended prior to mid-1977. Thus, program development is presumed to have started prior to 1974. No structured design was involved. The average fault density for module size classes is shown in Table 5-11. Size is expressed in source code statements.

TABLE 5-11.
EFFECT OF MODULE SIZE:
DATA SOURCE 1

Statements/Module	No. of Modules	Fault Density
< 200	24	.085
200 - 3,000	73	.025
> 3,000	10	.004

This shows a consistent trend of lower fault density with increasing module size. This is somewhat surprising in view of the emphasis in many recent software development specifications on small module size. Small modules are preferred for ease of maintenance and re-use. This data indicate that modules comprising less than 200 statements will carry a reliability prediction multiplier approximately 3 compared to "average" modules (this term here meaning between 200 and 3,000 statements), and that very large modules may carry a multiplier of 0.3 or less.





Simple - Y : FD X : SIZE

DF:	R-squared:	Std. Err.:	Coef. Var.:
48	.018	.011	117.706

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	.01	.002	.000003051	5.834
SLOPE	-.000006036	.000006441	4.148E-11	-.937

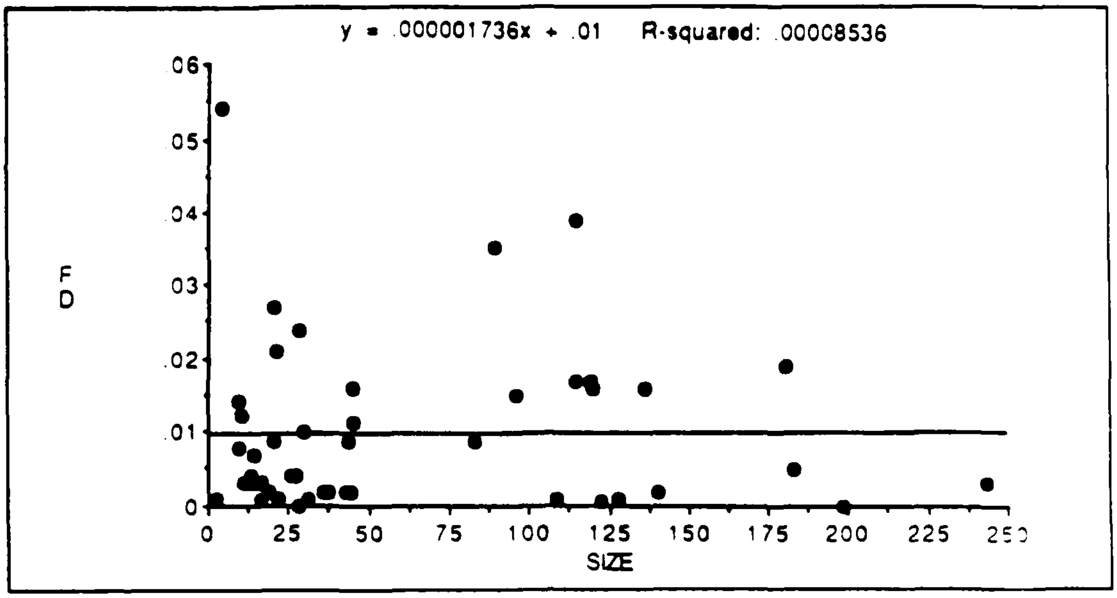
Analysis of Variance Table

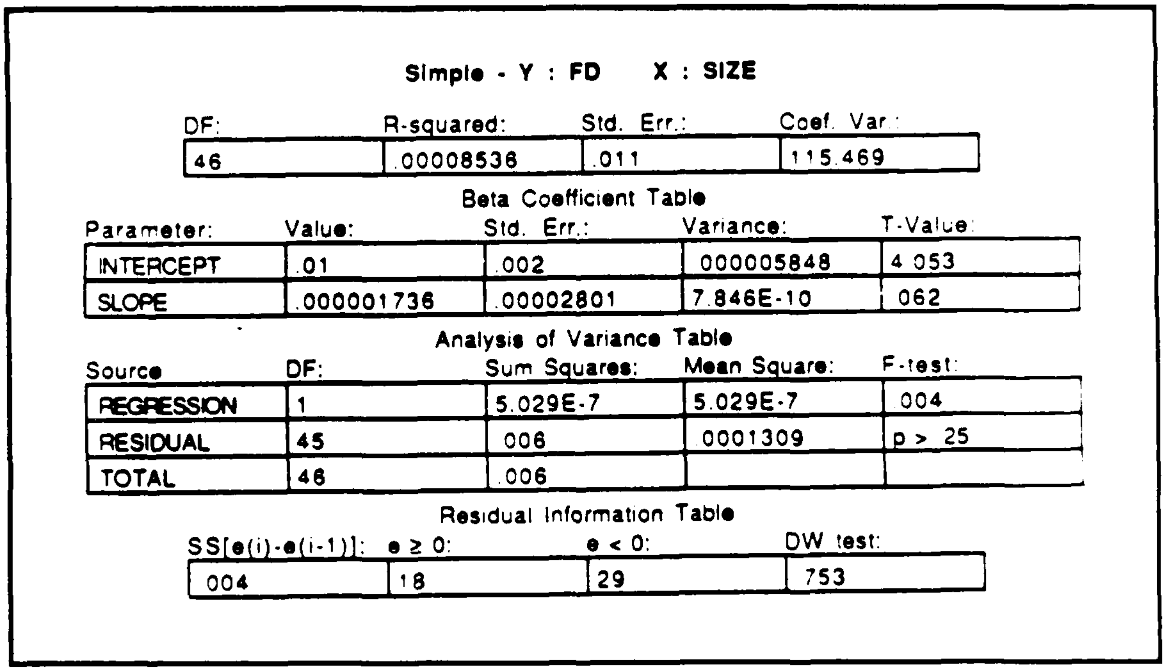
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	.0001109	.0001109	.878
RESIDUAL	47	.006	.0001262	p > .25
TOTAL	48	.006		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
.007	17	32	1.157

FIGURE 5-9 RELATIONSHIP OF SIZE TO FD





Simple - Y : FD X : SIZE

DF:	R-squared:	Std. Err.:	Coef. Var.:
46	.00008536	.011	115.469

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	.01	.002	000005848	4 053
SLOPE	.000001736	.00002801	7.846E-10	062

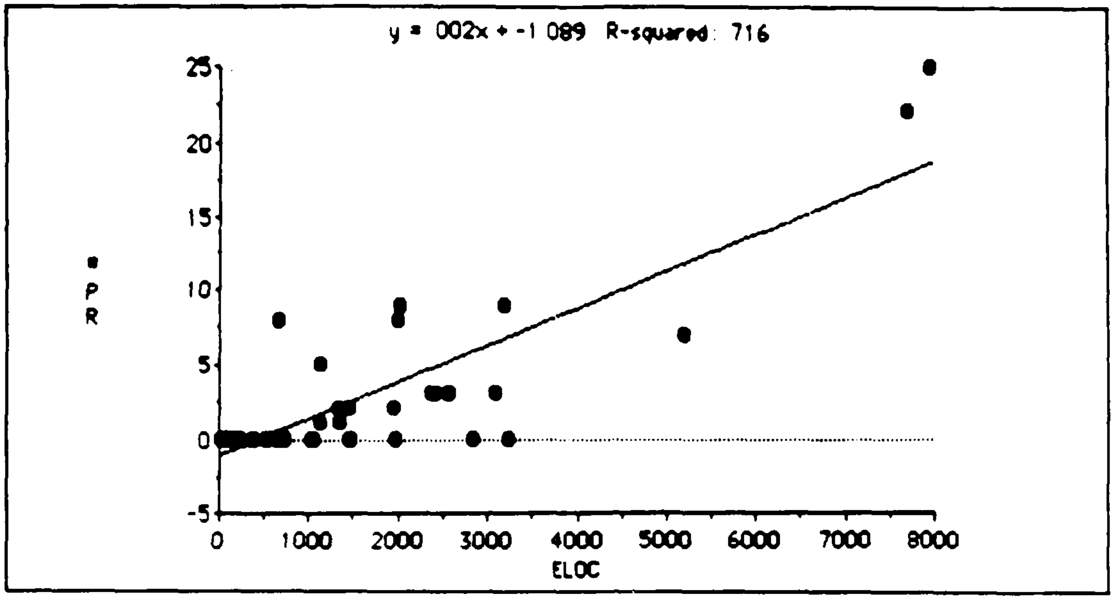
Analysis of Variance Table

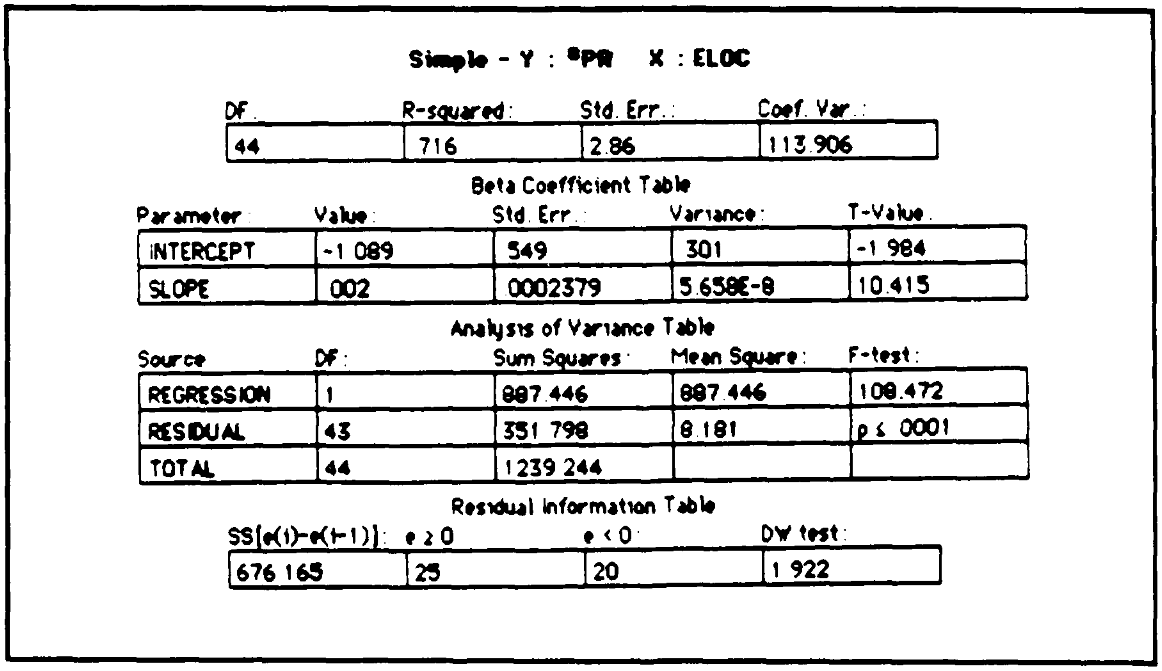
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	5.029E-7	5.029E-7	004
RESIDUAL	45	006	0001309	p > 25
TOTAL	46	.006		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
004	18	29	753

FIGURE 5-10 MODIFIED ANALYSIS OF SIZE VS FAULT DENSITY





Simple - Y : #PR X : ELOC

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	716	2.86	113.906

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-1 089	549	301	-1 984
SLOPE	002	0002379	5.658E-8	10.415

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	887.446	887.446	108.472
RESIDUAL	43	351 798	8.181	p ≤ 0001
TOTAL	44	1239 244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0	e < 0:	DW test:
676 165	25	20	1 922

FIGURE 5-11 SIZE VERSUS FAULT DENSITY
(DATA SOURCE 10)

Data collected during this effort is more intuitively supportive. Data collected from data sources 10 and 17 is in Table 5-12. Here units which were under 200 lines of code performed extremely well.

TABLE 5-12.
EFFECT OF MODULE SIZE:
DATA SOURCE 10, 17

No. of Processes	Executable Statements/Unit	Fault Density
3	< 50	0
3	50 < < 100	0
9	101 < < 200	0
15	TOTAL < 200	0
7	201 < < 999	.0017
10	1000 < < 1999	.0014
12	2000 <	.0015
29	TOTAL > 2000	.0015

Data available from data source 11 is shown in Table 5-13.

TABLE 5-13.
EFFECT OF MODULE SIZE:
DATA SOURCE 11

Statements/Module	No. of Modules	Fault Density
< 100	23	.094
100 - 1,000	4	.044
> 1,000	1	.047

In [GRAS82], a relationship between module size and number of problem reports was found to be:

$$PR's = .012\ S - 9.3$$

where PR's = Number of Problem Reports
and S = Number of Lines of Code

In [MOTL76], the relationships shown in Figure 5-12 were developed.

The obvious conclusion is that no consistent relationship could

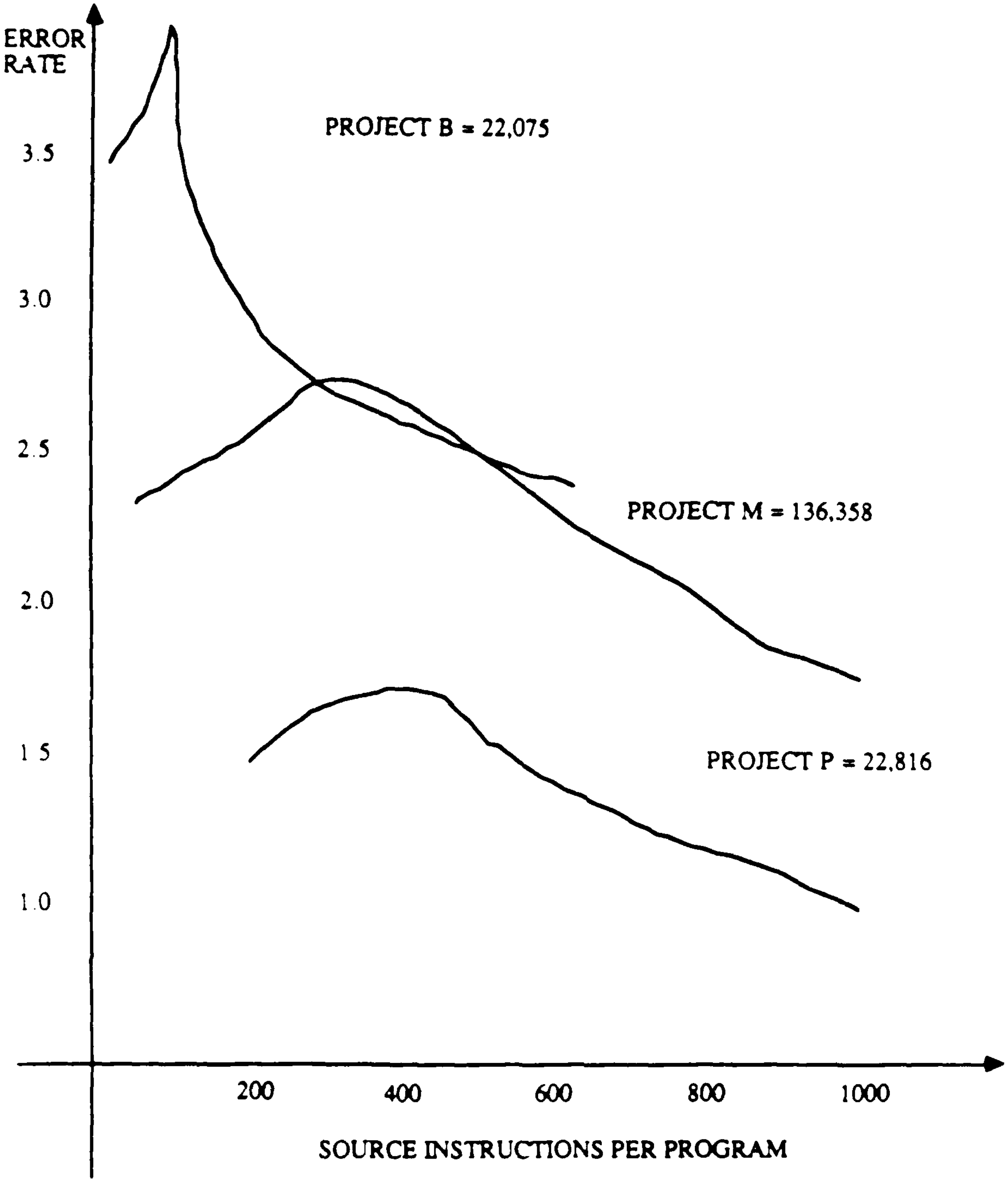


FIGURE 5-12 ERROR RATE AND SOURCE INSTRUCTION RELATIONSHIP FOR [MOTL 76]

be derived. Within an organization or a project team it appears there may be some consistency which would lead to analyses of the impact of standards and methodologies on module size and fault density. This type of analysis was not done.

In spite of this finding, a metric which reinforces the standards typically found in software development organizations was developed. That metric recognizes benefits of small modules (shown in analyses of data sources 10 and 17) by assigning a multiplier of .9 to modules less than 200 LOC and recognizes inherent difficulties with extremely large modules (›3,000 LOC) by assigning a multiplier of 2 to these large modules. All other modules are assigned a multiplier of 1. The overall multiplier recommended is a weighted average based on the number of modules in each category.

Complexity

For data source, [WILL77], a classification of modules into "simple", "medium", and "complex" was available. It is stated that the assignment of these attributes was made without firm criteria, but that "no difficulty was encountered in assigning complex or simple to a module".

The overall fault densities for each of the complexity categories are shown below:

Simple	.026
Medium	.013
Complex	.029

Because of the inconsistency in this effect, and possible compounding the affect of language and size with complexity, more detailed analyses were performed as indicated in Table 5-14.

TABLE 5-14.
EFFECT OF COMPLEXITY FOR SUBCLASSES OF CODE

Complexity Designation	Fault Density, Percent			
	Subclass:			Size:
	Assembly	Mixed	JOVIAL	200-999
Simple	0.1	0.5	3.4	3.5
Medium	0.3	3.4	1.9	3.8
Complex	2.2	0.8	4.1	2.5

Only for pure assembly code, a subclass that includes relatively few modules, does the fault density exhibit the expected relation to complexity. In all other subclasses, the effect of complexity (as assessed here) on fault density seems to be random.

A subjective evaluation of complexity as "easy", "medium", and "hard" is also provided in the SEL Component Summary Form, but no analysis of that information relative to fault density was performed since it was assumed it would not provide conclusive data.

Use of the data collected in Table 5-4 for Data Sources 10 and 17 to quantitatively calculate a complexity metric based on the McCabe cyclomatic complexity metric and relate that to fault density exhibited better results. Figure 5-13 illustrates the results of the regression analysis using the McCabe complexity metric for data source 10 and 17. The relationship illustrated here is:

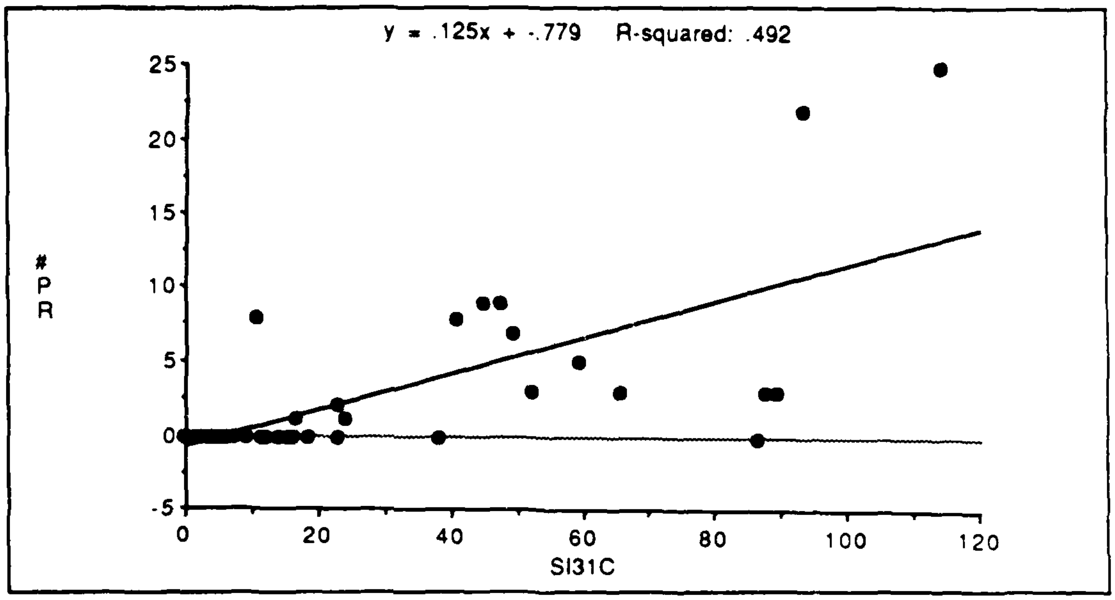
$$FD = -.009\ C + .001$$

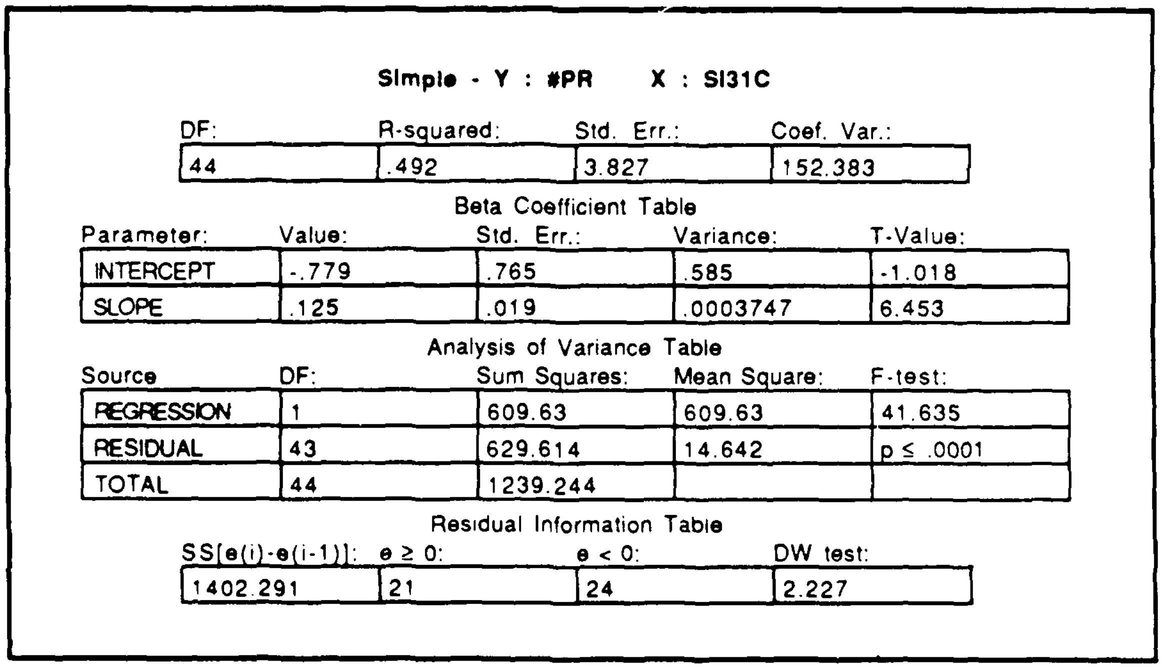
The negative slope is consistent with the way we have defined the complexity metric, i.e. as the metric approaches zero complexity increases. The correlation coefficient is not supportive of using the above relationship generally. What is apparent from the plot of data, however, is that the processes with a McCabe's metric greater than .05 (which is a cyclomatic complexity of 20) are more likely to be these procoesses with a higher fault density. Based on this observation, a multiplier of 1.5 is recommended for modules with a complexity greater than 20, 1 for modules with a complexity between 7 and 20, and .8 for those modules with a complexity less than 7. The overall multiplier will be a weighted average of those scoores by the number of modules in each category.

Standards Review

The Standards Review represents code inspections, walkthroughs or standard enforcement results. In Table 5-4 there are a number of data elements which make up the Standards Review Checklist described in Volume II. Figures 5-14 through 5-19 illustrate the correlations found between various measurements/elements and the number of problems found in a process. The ones illustrated in these figures are:

Figure 5-14:	S148C	- a function of the number of Branches/ELOC)
Figure 5-15:	S146C	- a function of the number of Statement Labels/ELOC)
Figure 5-16:	LOOPS	- number of Loops
Figure 5-17:	NEST_DEPTH	- Maximum Nesting Depth Level
Figure 5-18:	DATA_MANIP	- number of Data Manipulation Statements
Figure 5-19:	DATA ITEMS	- number of Data Items





Simple - Y : #PR X : SI31C

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.492	3.827	152.383

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-.779	.765	.585	-1.018
SLOPE	.125	.019	.0003747	6.453

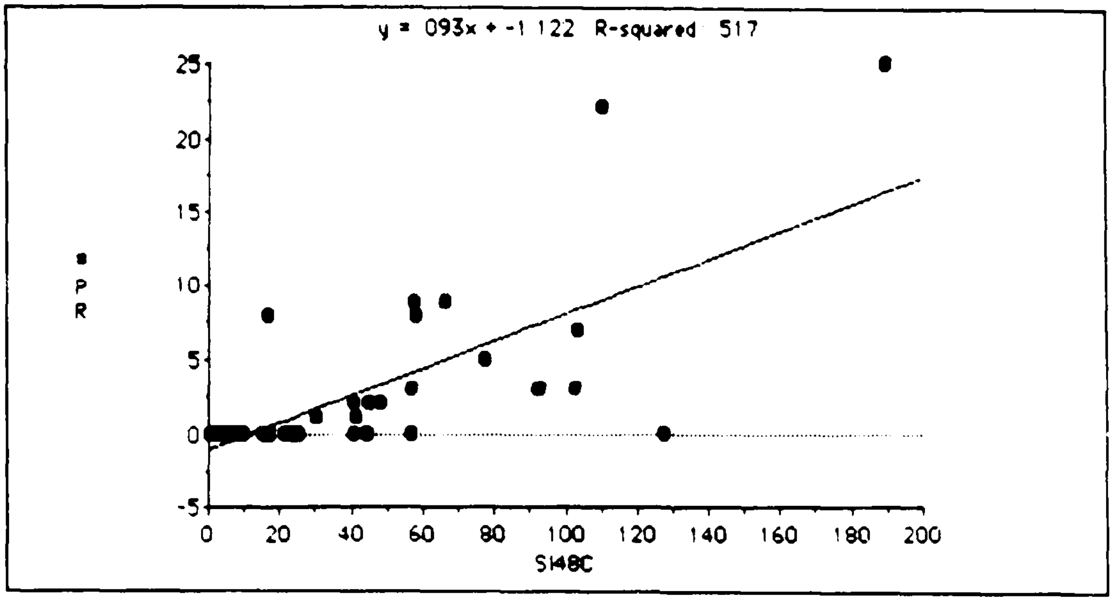
Analysis of Variance Table

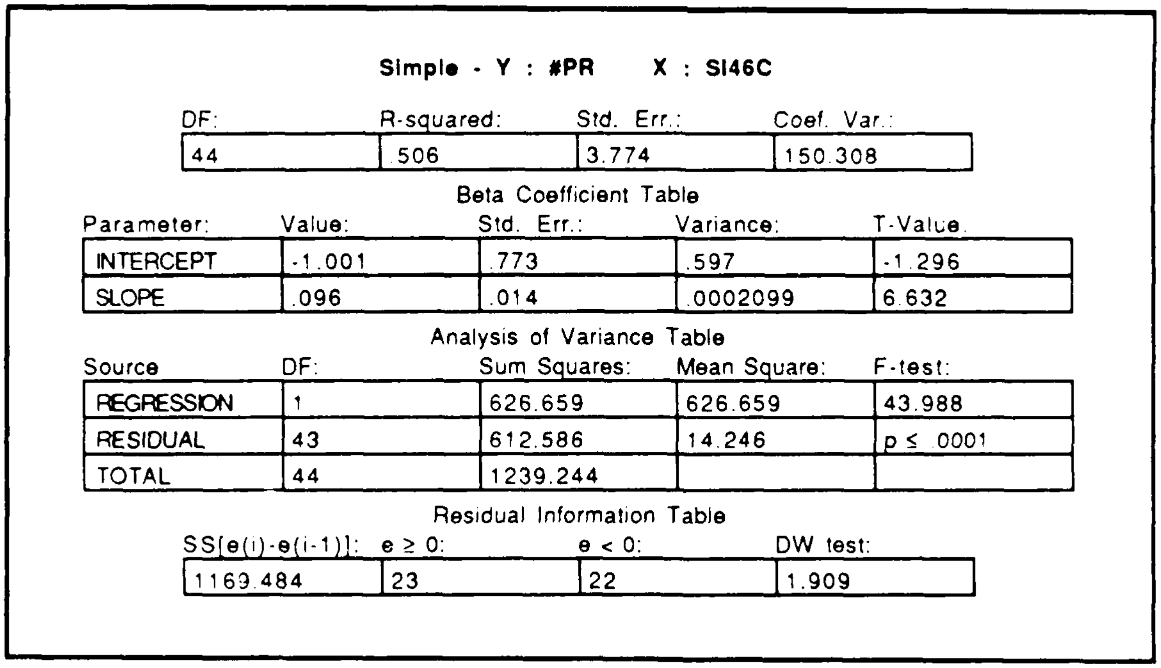
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	609.63	609.63	41.635
RESIDUAL	43	629.614	14.642	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
1402.291	21	24	2.227

FIGURE 5-13 REGRESSION ANALYSIS USING MCCABE COMPLEXITY METRIC





Simple - Y : #PR X : SI46C

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.506	3.774	150.308

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-1.001	.773	.597	-1.296
SLOPE	.096	.014	.0002099	6.632

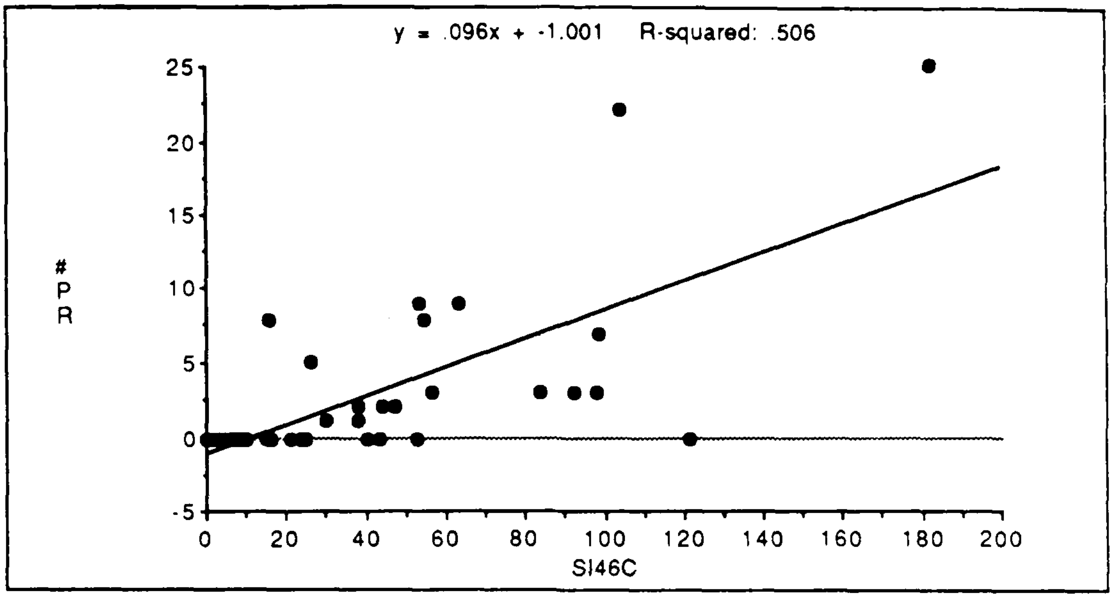
Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	626.659	626.659	43.988
RESIDUAL	43	612.586	14.246	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
1169.484	23	22	1.909

FIGURE 5-14 S148C



Simple - Y : #PR X : SI46C

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.506	3.774	150.308

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-1.001	.773	.597	-1.296
SLOPE	.096	.014	.0002099	6.632

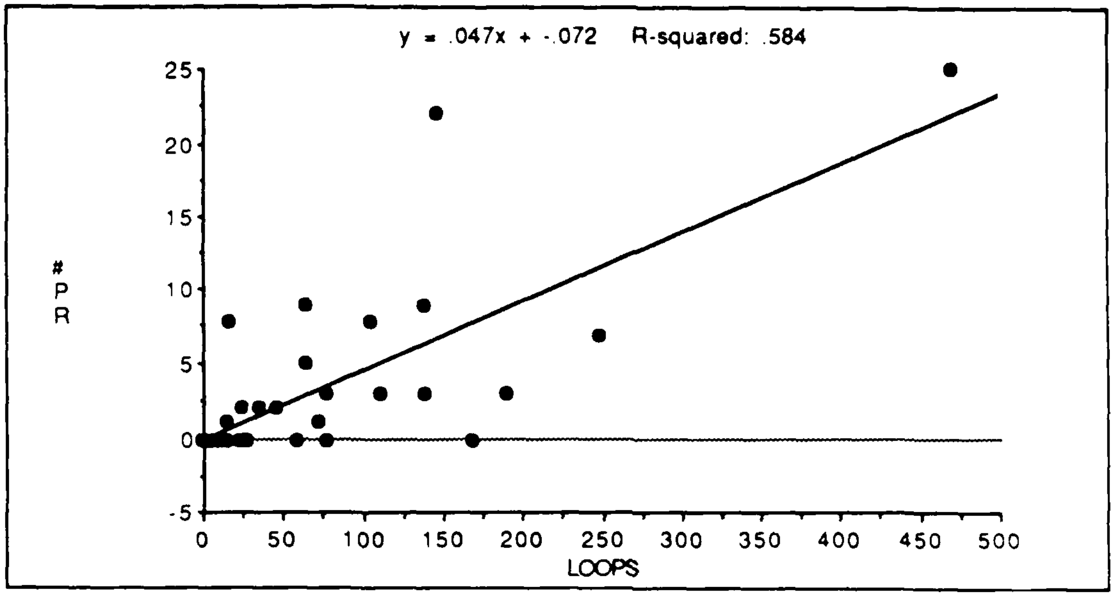
Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	626.659	626.659	43.988
RESIDUAL	43	612.586	14.246	$p \leq .0001$
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
1169.484	23	22	1.909

FIGURE 5-15 S146C



Simple - Y : #PR X : LOOPS

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.584	3.463	137.892

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-.072	.614	.377	-.117
SLOPE	.047	.006	.00003615	7.769

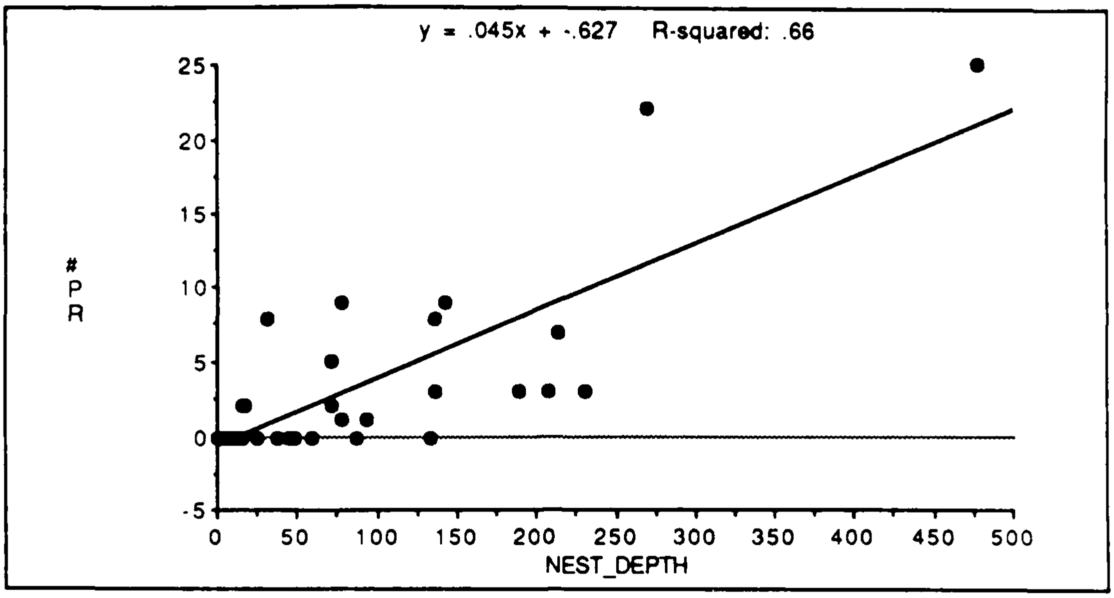
Analysis of Variance Table

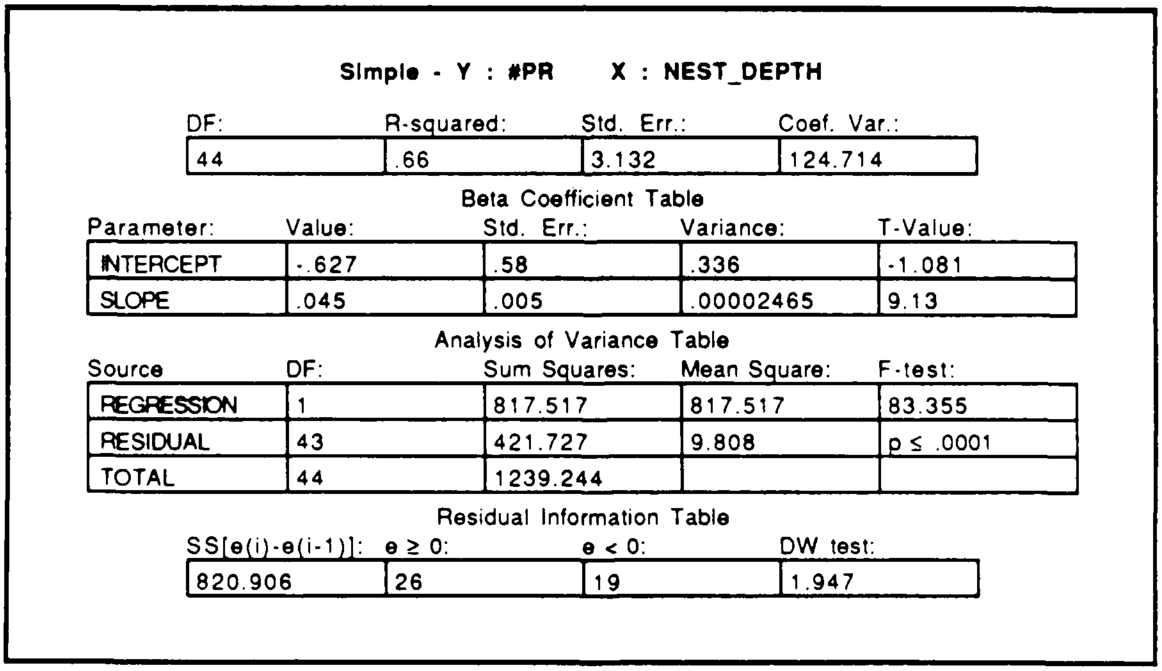
Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	723.687	723.687	60.359
RESIDUAL	43	515.558	11.99	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
1052.445	16	29	2.041

FIGURE 5-16 LOOPS





Simple - Y : #PR X : NEST_DEPTH

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.66	3.132	124.714

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-.627	.58	.336	-1.081
SLOPE	.045	.005	.00002465	9.13

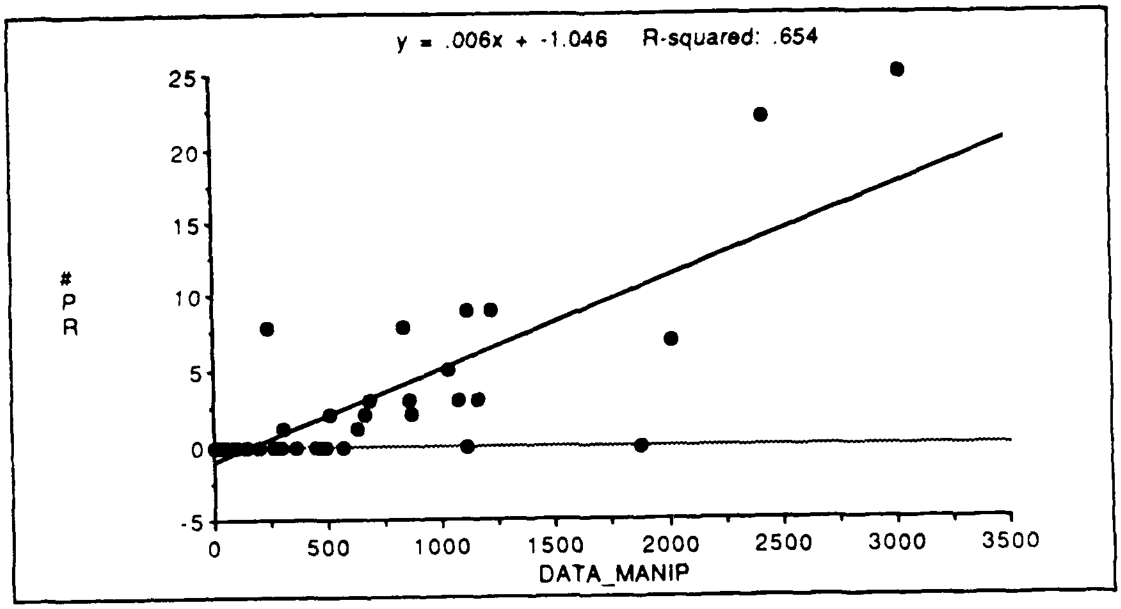
Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	817.517	817.517	83.355
RESIDUAL	43	421.727	9.808	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
820.906	26	19	1.947

FIGURE 5-17 NEST_DEPTH



Simple - Y : #PR X : DATA_MANIP

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.654	3.157	125.726

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-1.046	.614	.377	-1.703
SLOPE	.006	.001	4.721E-7	9.018

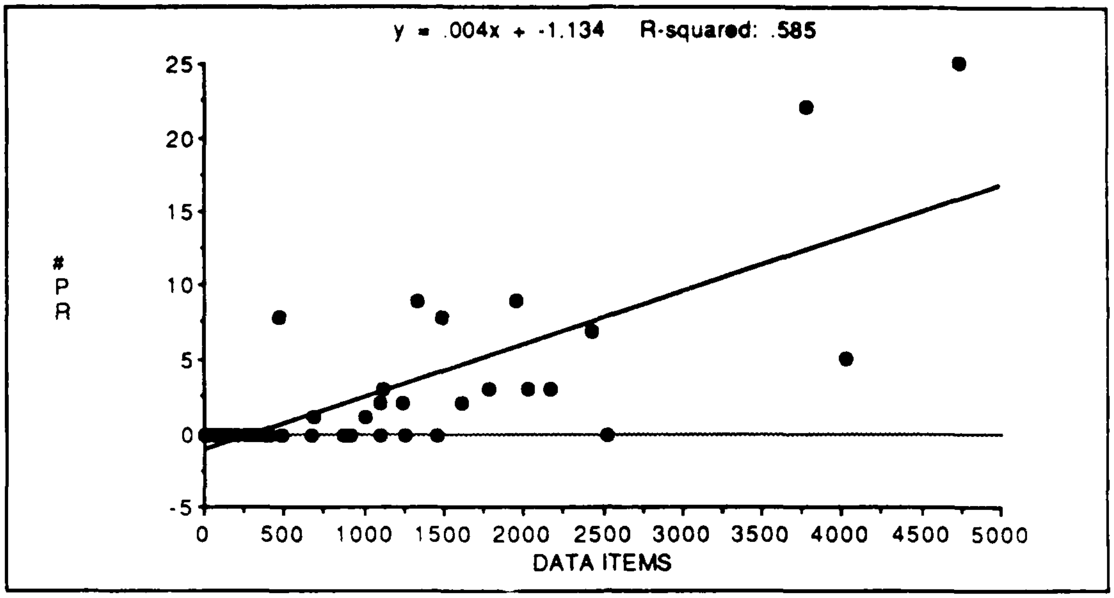
Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	810.649	810.649	81.33
RESIDUAL	43	428.596	9.967	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
920.24	25	20	2.147

FIGURE 5-18 DATA_MANIP



Simple - Y : #PR X : DATA ITEMS

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.585	3.46	137.788

Beta Coefficient Table

Parameter:	Value:	Std. Err.:	Variance:	T-Value:
INTERCEPT	-1.134	.697	.486	-1.627
SLOPE	.004	.0004618	2.133E-7	7.779

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	1	724.46	724.46	60.514
RESIDUAL	43	514.785	11.972	p ≤ .0001
TOTAL	44	1239.244		

Residual Information Table

SS[e(i)-e(i-1)]:	e ≥ 0:	e < 0:	DW test:
1218.992	24	21	2.368

FIGURE 5-19 DATA ITEMS

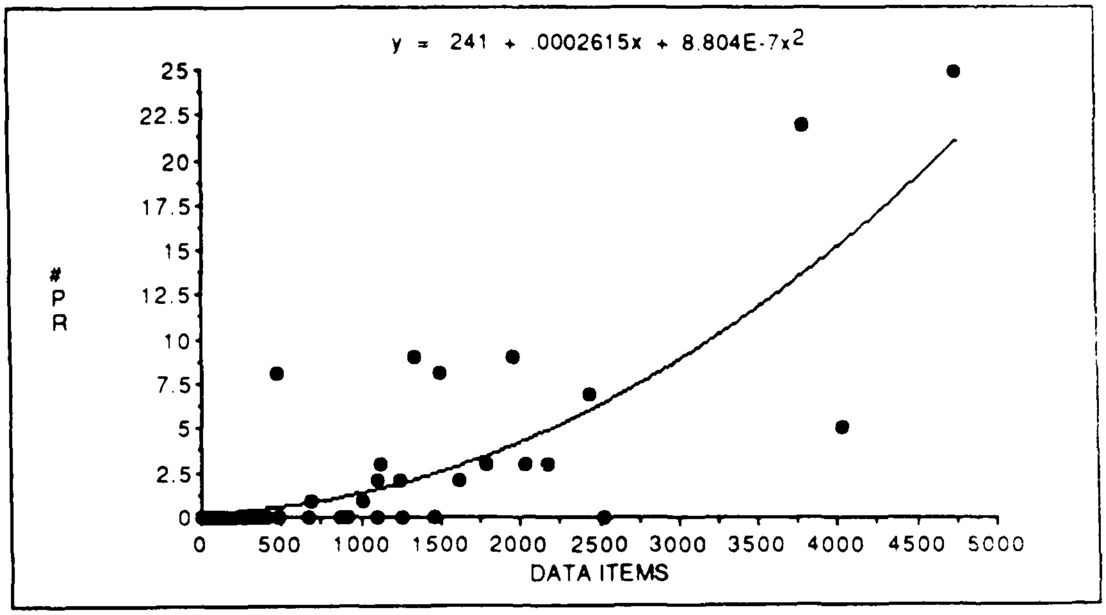
Figure 5-20 illustrates a non-linear regression analysis. This is the same metric and data as shown in Figure 5-19. The non-linear regression analysis resulted in a slightly better fit.

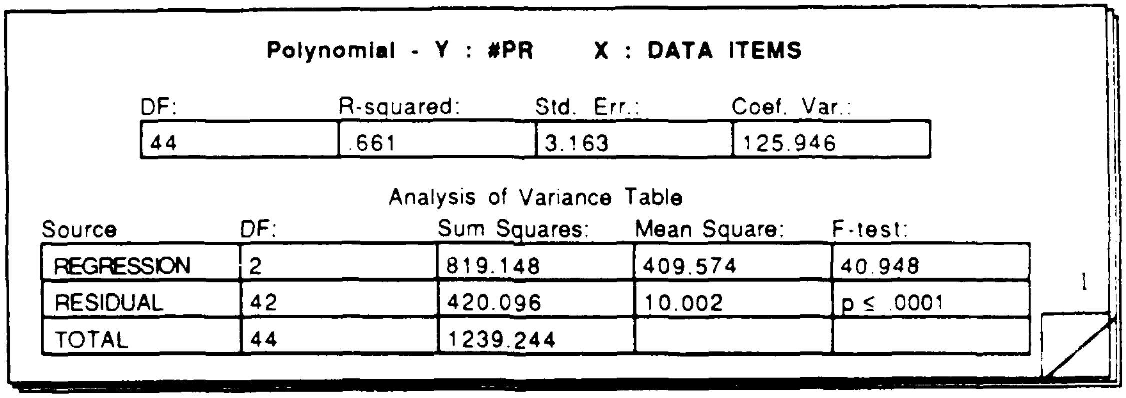
The regressions were calculated using number of problem reports and fault density. Better correlations were found with number of problem reports as the independent variable and those analyses are presented here. We found in data sources 10 and 17 that over 60 percent of the processes (CSC's) had no problems reported against them. Only 15 percent had more than 3 problems written against them which based on the average size of a process equated to a fault density greater than .0015.

A key use then of these metrics for improving S/W reliability is to pinpoint these problem modules for predictive purposes but primarily for identification and correction. As an illustration of this concept, using the metric, number of data items, to identify the potential problem modules, we flagged all processes that have more than the average number of data items (997). In retrospect, this technique would have identified 86 percent of the problem modules. The identification is not perfect, i.e. other modules were also identified by the metric that were not problem modules by our definition. But the predictive performance seems excellent. The results were:

- 42% of all processes flagged
- 84% of processes flagged had problems
- Identified 88% of all process with problems
- Identified 86% of all problem processes (those with fault densities higher than the average for the overall system).

For purposes of prediction, the metric recommended is based on the percentage of problem modules identified by the metrics. If over half of the modules are flagged as potential problem modules by the metrics applied as a standards review then the predicted reliability should be raised since the expected problems seem manageable. In data sources 10 and 17, the problem processes had a fault density of .0035, twice the average fault density of the system, .0017. These problem processes accounted for 15 percent of the processes. Thirty-eight (38) percent of the processes had problems with an average fault density of .0024, 1.4 times the average. For prediction purposes then, the following multipliers are recommended (Table 5-15):





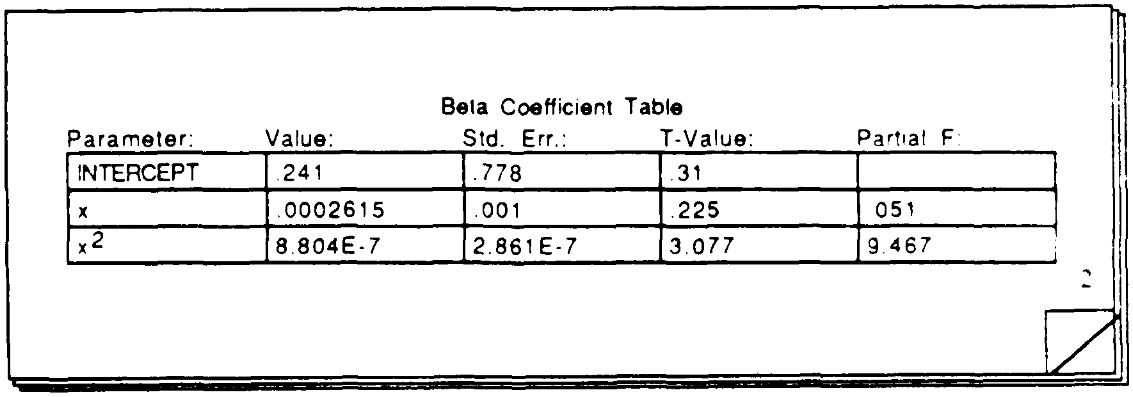
Polynomial - Y : #PR X : DATA ITEMS

DF:	R-squared:	Std. Err.:	Coef. Var.:
44	.661	3.163	125.946

Analysis of Variance Table

Source	DF:	Sum Squares:	Mean Square:	F-test:
REGRESSION	2	819.148	409.574	40.948
RESIDUAL	42	420.096	10.002	$p \leq .0001$
TOTAL	44	1239.244		

1



Beta Coefficient Table

Parameter:	Value:	Std. Err.:	T-Value:	Partial F:
INTERCEPT	.241	.778	.31	
x	.0002615	.001	.225	.051
x^2	8.804E-7	2.861E-7	3.077	9.467

2

FIGURE 5-20 NON-LINEAR REGRESSION ANALYSIS

TABLE 5-15
RECOMMENDED SR METRIC

Standards Review Metric (SR)	Percent of Modules Flagged as Potential Problems
1.5	› 50
1	50 to 25
.75	‹ 25

This approach is recommended based on the data observed in data sources 10 and 17. A larger sample is required to derive an actual prediction equation as described in Section 3.

5.2.4 Test Metrics

<u>TEST EFFORT</u>

Three data sources, 10 , 17 and 26, were used for demonstrating the Test Effort metric. Table 5-16 presents the data available from the three data sources.

TABLE 5-16.
TEST EFFORT VERSUS FAULT DENSITY/FAILURE RATE

Data Source	Test Effort	Fault Density	Failure Rate
10	8% of Dev	.0016	.18
17	10% of Dev	.002	.007
26 SYSTEMS A	12 MONTHS	.0075	not available
B	4 MONTHS	.007	"
C	8 MONTHS	.01	"
D	6 MONTHS	.0095	"

Additional data is needed to derive a generally useful relationship.

<u>TEST METHODOLOGY</u>

The Test Methodology metrics were calculated for data sources 10 and 17 (shown in Table 4-4). As shown in Table 5-17, the higher scoring test methodology is related to the lower fault density which is intuitive.

TABLE 5-17.
TEST METHODOLOGY METRIC VERSUS FAULT DENSITY

Data Source	Test Methodology	Fault Density
10	.67	.0016
17	.44	.002

TEST COVERAGE

No analysis was performed on Test Coverage.

FAILURE RATE TRENDS DURING TEST

Using the findings presented in Table 5-2 and Figure 5-2d, a multiplier of .2 can be used to estimate the failure rate at end of test based on the average failure rate observed. A multiplier of .14 can be used to estimate operational failure rate based on end of test failure rate.

5.2.5 Operational Estimation Metrics

Workload

Significant effects of workload on software failure rates have been reported by investigators at Stanford University [ROSS82]. The hazard function, the incremental failure rate due to increasing workload, ranges over two orders of magnitude. This indicates that the workload must be taken into account in arriving at software reliability predictions.

For military applications, workload effects can be particularly important. During time cf conflict, the workloads can be expected to be exceptionally heavy, causing the expected failure rate to increase, and yet at that same time a failure can have the most serious consequences. Hence, predictions of failure rates that do not take workload effects into account fail to provide the information that Air Force decision makers need.

The mechanism by which workload increases the failure rate is not completely known, but it is generally believed to be associated with a high level of exception states, such as busy I/O channels, long waits for disk access, and possibly increased memory errors (due to the use of less frequently accessed memory blocks). Data presented in [IYER81] show that the highest software (and also hardware) failure rates were experienced during the hours when the highest levels of exception handling prevailed.

Details of workload effects on software failure rate are still a research topic, and no specific work on a prediction function was performed as part of the present effort. Data from data source 10 substantiates the range of failure rates during operation. Table 4-5 and Figure 4-14 illustrated the fluctuation encountered. Discounting the spikes in Figure 4-14 (these represented installation of enhanced versions of the system) the range in problem reporting was 20 to 1 during operations.

The prediction function advocated is based on published work (see Figure 5-21 which is reproduced from [ROSS82]). The quantity plotted along the vertical axis is the inherent load hazard, z(x), defined as:

> Prob. of failure in load interval $(x, x+\delta x)$/Prob. of failure in interval $(0,x)$.

It measures the incremental risk of failure involved in increasing the workload from x to $x+\delta x$.

The horizontal axis shows three different measures of workload:

- Virtual memory paging activity, number of pages read per second (PAGEIN);
- Operating system overhead, fraction of time not available for user processes (OVERHEAD); and
- Input/output activity, number of non-spooled input/output operations started per second (SIO).

These graphs provide an option of predicting workload effects by any of the indicators of workload used here. The fraction of overhead usage is probably the most commonly obtainable quantity. From a practical point of view, before a computer installation becomes operational, the fraction of capacity to be used at maximum expected workload is probably the only indication of this factor that will be available early in the development.

In [TROY86], data sousrce 27, a function was developed relating software failures to user logins. That function:

$$y = 7.39 + 4.72 * 10^{-3} x$$

where y = number of software failures
and x = number of user logins

had a correlation coefficient of .44. The user logins could be viewed as an expression of workload.

Variability of Data and Control States

Software that is delivered for Air Force use is essentially fault

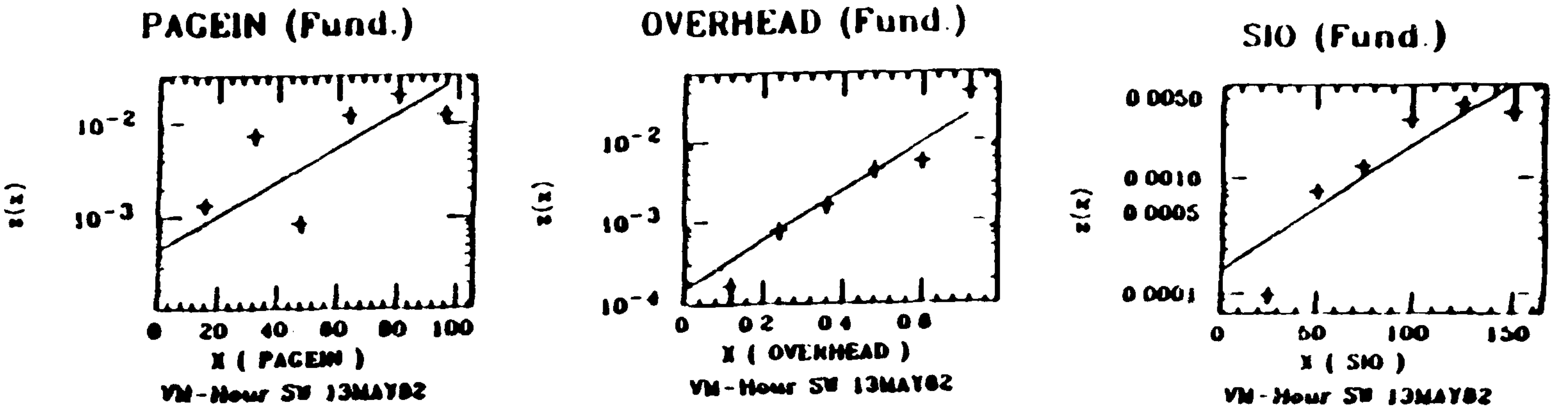


FIGURE 5-21 EFFECT OF WORKLOAD ON SOFTWARE HAZARD

free for nominal data and control states, i.e., where an input is called for, an input fully compliant with the specification will be present; when an output is called for, the channel for receiving the output will be available. A major factor in the occurrence of failures, and therefore affecting the failure rate, is the variability of input and control states and the abnormal data encountered.

Variability of the input data is the primary determinant of software reliability in some models, such as the ones proposed by Nelson and Lipow [DACS79] and Roger Cheung [CHEU81]. Neither one of these models is supported by sufficient data to permit direct evaluation of the effect of variability on failure frequency. Nelson and Lipow propose partitioning of the input data set, and an index of variability can then be derived from the number of partitions accessed during one time period or one run. This appears practical in only a very limited number of applications. Cheung uses the calling sequence as an indicator of variability, a somewhat more easily implemented measure, but still targeted primarily to a research environment. A major difficulty with these approaches is that guidelines for their implementation can be provided only for a narrow spectrum of software applications. The partitioning of input states differ vastly between an operational flight control program, a message forwarding protocol, or a scientific computation.

It is proposed to use the frequency of exception conditions as a practical measure of variability in the current effort. Exception states include:

- Page faults, input/output operations, waiting for completion of a related operation -- the frequency of all of these is workload-dependent and the effect on software reliability is discussed in the next section;

- Response to software deficiencies such as overflow, zero denominator, or array index out of range; and

- Response to hardware difficulties such as parity errors, error correction by means of code, or noisy channel.

The last two of these combined in the input variability modifier for the operating environment, EV. Data presented in [IYER81], illustrated in Table 5-18, indicates that approximately 1,000 exception conditions of the latter two types were encountered in 5,000 hours of computer operation. A value of 0.2 exception conditions per computer-hour has therefore been adopted as the baseline, to be equated to unity. Because failures may arise even if no exception conditions at all are encountered, it is desirable to bias the modifier to a small positive value. A suggested form is

$$EV = 0.1 + 4.5E$$

where E is the number of exception conditions per hour. For E = 0.2, EV = 1.

In [TROY86], a function was derived relating software failures to hardware failures. That function:

$$y = 2.943 + .7189\ x$$

where y = number of software failures
and x = number of hardware failures

had a fairly good correlation coeeficient of .7. The hardware failures are obviously a form of exception conditions which [IYER83] related to software failures.

TABLE 5-18. SUMMARY OF EXCEPTION CONDITIONS
FOR AN IBM 3801 [IYER83]

	HARDWARE DETECTED		SOFTWARE DETECTED		ALL
ERROR TYPE	Freq.	%	Freq.	%	%
STORAGE MANAGEMENT	11	1.9	395	44.2	26.2
STORAGE EXCEPTIONS	382	67.0	0	0.0	24.7
DEADLOCKS	0	0.0	310	34.6	20.2
I/O & DATA MANAGEMENT	45	7.9	116	13.0	10.5
PROGRAMMING EXCEPTIONS	114	19.9	0	0.0	7.4
CONTROL	18	3.2	50	5.6	4.4
INVALID	1	0.1	23	2.6	6.6
ALL	57	100.0	894	100.0	100.0

5.2.6 Other Analyses

The data collected afforded additional analyses opportunities. For example, data about the types of problems reported were available from data sources 1, 2, 3, 4, 5, 16, 27, 28, 29, and 31. The fault categorization scheme used was originally presented in [THAY76] and is the most widely used scheme in the industry. Table 5-19 presents the data by data source and in summary form.

Table 5-20 provides a breakdown by functional category for four

TABLE 5-19 ERROR CATEGORIZATION DATA

PROBLEMS/ APPLICATIONS	1 STRATEGIC	2 AIRBORNE	3 STRATEGIC	4 STRATEGIC	5 STRATEGIC	17 STRATEGIC	16 STRATEGIC	28 STRATEGIC	32 STRATEGIC	33 DEVELOPMENTAL	AVERAGE
COMPUTATIONAL	5 1 (115)	5 4 (109)	7 6 (342)	9 0 (180)	3 2 (170)		5 0 (48)	17 0 (23)	10 5 (162)	12 0 (81)	6 0 (1232)
LOGIC	17 6 (382)	31 2 (635)	21 4 (960)	26 0 (521)	18 9 (993)	67 0 (836)	49 0 (462)	30 5 (41)	17 0 (256)	25 0 (169)	25 6 (5255)
I/O	1 0 (21)	1 4 (28)	16 2 (727)	16 4 (329)	8 6 (454)		2 0 (17)	3 0 (4)	10 4 (156)	8 0 (54)	8 7 (1790)
DATA HANDLING	18 9 (409)	13 4 (272)	13 5 (605)	18 2 (365)	6 7 (347)		38 0 (351)	7 0 (8)	11 0 (164)	11 0 (76)	12 6 (2598)
OS/SYS SPT	2 (4)	4 (8)	0 (1)		3 (14)		6 0 (57)	1 5 (2)	1 (1)		4 (87)
CONFIGURATION	8 (18)	6 (12)	1 8 (83)		4 (19)	17 0 (211)			1 0 (16)		1 7 (359)
MOD/MOD I/F	7 (17)	2 0 (41)	5 5 (248)	17 0 (341)	2 3 (123)			4 0 (6)	4 5 (69)	7 0 (48)	4 3 (892)
MOD/SYS I/F	8 (17)	2 (3)	7 (30)		7 (38)			5 0 (7)	2 (5)		5 (100)
TAPC PROC		3 (5)	2 (8)		1 (5)				1 (1)		1 (19)
USER I/F	5 (13)	6 (12)	8 5 (385)		5 (29)			3 0 (4)	7 5 (114)		2 8 (554)
DB I/F	1 5 (32)	8 (17)	5 (22)	4 1 (82)	3 4 (176)				6 4 (10)	16 0 (112)	2 2 (451)
USER CHANGES	35 3 (764)	7 9 (161)			3 6 (188)	7 5 (92)		13 0 (18)	7 4 (111)		7 1 (1445)
DATA SET	7 5 (162)	3 3 (67;	11 2 (501)	8 (16)	5 9 (310)			1 5 (2)	3 5 (56)	9 0 (61)	5 7 (1175)
GLOBAL VAR DEFN	2 1 (45)	2 3 (46)	1 2 (55)		2 2 (112)			1 0 (1)	1 5 (24)		1 4 (283)
RECURRENT	1 8 (39)	7 3 (148)	1 7 (78)		15 6 (820)			1 0 (1)	11 5 (172)		6 1 (1258)
DOCUMENTATION	7 (15)	1 3 (27)	4 2 (187)		15 3 (796)			11 0 (15)	6 4 (10)		5 1 (1050)
REQUIREMENTS COMPLIANCE	5 (10)	7 1 (144)	6 (26)	8 5 (171)	6 (32)			1 5 (2)	5 5 (85)	12 0 (86)	2 7 (556)
UNIDENTIFIED	3 7 (77)	5 5 (111)	2 2 (98)		6 4 (338)				5 5 (86)		3 5 (710)
OP	7 (15)	7 8 (158)	3 0 (134)		5 (26)	8 5 (106)					2 1 (439)
HARDWARE	5 (11)	1 6 (32)			4 7 (246)						1 4 (289)

JMB/12/86111

* Numbers shown are percent of total

** Numbers in parentheses are absolute numbers, ie total number of problem reports

TABLE 5-20 FUNCTIONAL DISTRIBTUION OF PR'S

DATA SOURCES

FUNCTIONS	1	2	10	16	AVERAGE
EVENT CONTROL		24	52	33	**27.2**
PROCESS CONTROL					**--**
PROCEDURE CONTROL					**--**
MSG PROC					**--**
SIGNAL PROC	9.6				**2.4**
PATTERN RELOG					**--**
EXEC/OS	9.5	13	6		**7.1**
SPT SW	10.3	29		5	**11**
RES MGT	1.3			16.5	**4.5**
SCIENTIFIC PROC	18.7				**4.7**
DECISION AID					**--**
DATA MGT	38.2			28	**16.5**
DIST'N/COMM	7.4				**2**
DISPLAY	5	6	42	17.5	**17.6**
DIAGNOSTICS		28			**7**

*Numbers shown are percent

data sources. Eventually failure rates for these functional categories of software should be sought to assess differences in failure rate at this level of detail.

Table 5-21 illustrates the fact that a small percentage (6) of the problems found are of a highly critical nature. Five systems were used to collect these data. Almost half of the problems reported are low criticality.

These additional analyses provide data to which future projects can be compared.

5.3 RESULTS OF ANALYSIS

The analyses performed using the 59 systems provided significant insight into software reliability. The data base created will provide an excellent basis from which to expand and further refine the relationships developed during this study. The immediate results were somewhat mixed. Tables 5-22 and 5-23 summarize the results. Table 5-22 illustrates our expectations (documented in Section 3) for each metric and what was realized (described in Section 5). The fact that specific statistically valid relationships were not derived for many of the metrics suggests one of the following:

(1) There isn't a relationship and the metric should not be used

(2) Our sample size was too small

(3) Some refinement in the metric is needed

The use of multipliers based on a table look up is dissappointing from a theoretical viewpoint because specific relationships were the goal of the research. Yet the table look up approach is based on observed relationships from data collected therefore represents the perceived impact on reliability.

The metrics recommended for use based on this analysis are indicated in Table 5-23. In all cases, further data collection and analysis would be beneficial. The available metrics are documented in a Guidebook (Volume II) to facilitate their application as software reliability predictors and estimators.

TABLE 5-21 SEVERITY OF PROBLEMS (PERCENT)

PR CRITICALLY	DATA SOURCES					AVERAGE
	1	10	29 (2 Systems)		31	
HIGH	6	15.5	7	1	1	6
MED	76	53	50	56	15	50
LOW	18	31.5	43	43	84	44

Numbers shown are percent of total

TABLE 5-22 SUMMARY OF ANALYSIS

METRIC	EXPECTED FORM OF RELATIONSHIP (SECTION 3)	CURRENT RECOMMENDED APPROACH BASED ON DATA (SECTION 5)
Application (A)	Table of Average Fault Densities by Category	Table of Average Fault Densities by Category
Development Environment (D)	$D_L * D_C$	$D = D_O$ where D_O = 1 3 (E) 1 (S) 76 (O) or D_M = (.109 D_C - .4)/.014 (E) (.008 D_C - .003)/.013 (S) (.018 D_C - .003)/.008 (O) where D_C = Checklist Score between 0 and 1 restrict range of D_M to .5 to 2
Anomaly Management (SA)	ka/Am	SA = .9 if AM > .6 1 if .4 < AM < .6 1.1 IF AM < .4
Traceability (ST)	kt/TC TC=NR/(NR DR)	ST = 1.1 if (NR-AR)/NR< 9 1 if (NR-AR)/NR ≥ 9
Quality Review (SQ)	kq (NR/(NR-NDR))	SQ = 1.1 if DR/NR > .5 1 if DR/NR ≤ .5
Language (SL)	%Hol + 1.4% AL	SL = 1 (%HOL) + 1.4 (%AL)
Size (SS)	Ss(1) if LOC ≤ 10K Ss(2) if 10K LOC ≤ 50K Ss(3) if 50K LOC ≤ 100K Ss(4) if 100K < LOC	No Relationship found
Modularity (SM)	Sm(1) if M ≤ 200 Sm(2) if 200 < M ≤ 3000 Sm(3) if 3000 < M	SM = 9 u + w + 2x where u is no. of mods < 200 w is no. of mods between 200 and 3000 x is no. of mods > 3000
Reuse (SU)	SU(i) for % of revised code	No Relationship Found
Complexity (SX)	kx ∑ Sx(i)/n	Sx = 1.5a + b + .8c where a is no. of mods with C ≥ 20 b is no. of mods 20 > C ≥ 7 c is no. of mods C < 7
Standards Review (SR)	kr (n/cn PR)	SR = 1.5 if PR/NM ≥ .5 1 if 5 > PR/NM ≥ 25 .75 if PR/NM < 25

Test Effort (TE)	40/AT
	or
	TT (i)
		TE = .9 if 40 AT ≤ 1
		otherwise = 1

Test Methodology (TM)	ktc * TT/TU	TM = .9 for TT/TU ≥ .75 1 for .75 > TT/TU ≥ .5 1.1 for TT/TU < .5
Test Coverage (TC)	ktc/VS	TC = 1/VS
Workload (EW)	kew • ET/(ET-0S)	EW = ET/(ET-0S)
Input Variability (EV)	.1 + 4.5 • EC	EV = .1 + 4.5 EC

TABLE 5-23 RECOMMENDED METRICS

METRIC	ANALYSIS PERFORMED		RELATIONSHIP	
	SYSTEM LEVEL	DETAILED LEVEL	AVAILABLE	RECOMMENDED
Application	•		•	•
Development Environment	•		•	•
Anomaly Management		•	•	
Traceability	• (See Section 6)		•	
Quality Review		•	•	
Language	•		•	•
Size	•			
Modularity		•	•	
Complexity		•	•	•
Standards Review		•	•	•
Test Effort	•		•	
Test Methodology	•		•	
Test Coverage	• (See Section 6)		•	•
Workload	•		•	•
Input Variability	•		•	•

6.0 EXPERIMENTAL APPLICATION AND ASSESSMENT

6.1 Experiment

In order to assess the approach that was derived during this project, an experiment was conducted. That experiment involved the application of the prediction and estimation techniques identified in the preceeding Sections of this report and described in Guidebook format in Volume II. Those techhniques were applied to a development effort. In order not to bias the results, the application of the techniques was performed in line with the development effort but feedback was not given to the project team.

The development effort was to develop the Facilities Automated Maintenance Management/Engineering System (FAMMES) which performs work order processing (WO), Preventive Maintenance Scheduling (PM), Inventory Control (IC), and provides a maintenance history (MH) data base. The users of this system are Air Force maintenance personnel including supervisors, schedulers, analysts, and maintainers. The hardware architecture involved a DEC MicroVAX II, Rainbow Intelligent workstations, and VT100 terminals. System software utilized included a relational data base management system, a forms management system, an on-line query capability, and a code management system. The application software was written in FORTRAN.

The development of an initial operating capability was performed by a small team over a 3 month period and then incremental enhancements were made over 3 more months. Development testing was performed over a two month period, IOT&E/Acceptance testing was performed at the customer site, and the customer used the system over a 6 month period, reporting any problems encountered.

Table 6-1 provides summary statistics of the application code. The system was 16K lines of executable source code. The metrics provided in this table, eg. %I/O and complexity, are average values for the modules in each of the subsystems.

The problem report data collected is shown in Table 6-2.

The significant data collection performed for this study was in the area of test data. Table 6-3 provides a time series listing of all testing performed on the system. It includes developmental testing, on-site installation and training, preparation for the acceptance test, and acceptance testing and IOT&E by the customer and operational experience. The columns in this table show each test run, a users manual reference if the test was demonstrating a user function, problem reports generated per test run, what subsystem the problem was reported against, the cause of the failure according to the scheme in the legend, a classification of the impact of the failure and the time to fix, as well as the CPU time and wall clock time recorded for each

TABLE 6-1
SAMPLE SOFTWARE CHARACTERISTICS

SUB SYSTSEMS	TOT LOC	ELOC	%C	%I/O	MAX NEST	COMPLEXITY	HALSTEAD L	# MOD	% USE OF SYS SW
PM	5940	3062	.77	.03	2.36	.18	107	83	.28
MH	589	246	.65	.05	2.7	.16	166	6	.31
SYS	2529	1152	.93	.02	1.7	.19	93	45	.20
WO	8872	6680	1.2	.01	2.5	.18	160	144	.27
IC	7517	4956	1.0	.02	2.5	.16	119	133	.30
TOTAL	25447	16096	.99	.02	2.4	.185	133	411	.25

Table 6-2.
Discrepancy Report Data Collected

- **DURING DESIGN REVIEW**

    CRITIQUES OF DESIGN MATERIAL

- **DURING DEVELOPMENT TESTING**

    TEST CASE NUMBER
    IMPACT OF FAULT
    COST TO REPAIR
    CAUSE
    MODULE EFFECTED
    TEST TIME

- **DURING INSTALLATION, ON-SITE TRAINING, AND CUSTOMER IOT & E/ACCEPTANCE TEST**

    PROBLEM DESCRIPTION
    TYPE
    EFFECT
    CRITICALITY LEVEL
    METHOD OF DETECTION
    RECOMMENDED SOLUTION

TABLE 6-3
FAMMES TEST RESULTS (DEVELOPMENT TESTING)

TEST RUN	UM REF	PROBLEM REP	SUBSYS	CAUSE	IMPACT	FIX	CPU TIME (SECONDS)	COMPUTER TIME
Development Testing								
PY 01		–	IC				3.04	4 42.62
2	4.1.1	1	IC	A3g	M	L	4.13	19.48.22
		2	IC	A2b	L	M	–	–
3	4.1.3	–	IC				3.07	3:04.72
4	4.1.2	1	IC	A3e	L	L	2.85	5:03.35
		2	IC	A3d	L	L		
		3	IC	A3g	L	L		
5	4 2 2	–	IC				3.03	6.18.41
6	4.4						3.20	7:07.33
7	4.15		IC				7.85	8:53.60
8	4.5.3						3.01	14:54 01
9	4 1 3	1	IC	A2b	L	L	2.98	12:38.47
10	4 1.7		IC				3.06	10:42.36
11	4 1.8 4 1.3						3.04	10.24.44
12			PM				2.86	10.21.29
13		1	PM	AIC	L	L	2.90	12:35.16
14			PM				2.98	34:24.63
15			PM				3 21	6:42.74
20			PM				2 98	7:15 59
21	4 1.8,	1	IC	AIC	M	L	3 42	24:23 77
22	4 1 3	1	IC	A2b	H	L	3 63	17 47 80
23	4 2 1		IC				3 08	3 23 12
24	3 4		IC				23 56	6 24 70
25	1.4		PM				3 76	11 26 63
26		1	WO	A2b	H	M	15 76	6 14 65
27	1 2		PM				3 03	3 19 43
28	4 16		MH				2 91	14 14 15
29	3 6	1	WO	A2c	L	L	4 06	22 15 03
		1		Ab	L	M		
30	4 4 1		IC				2 92	2 42 55

TABLE 6-3 (Continued)
FAMMES TEST RESULTS (DEVELOPMENT TESTING)

TEST RUN	UM REF	PROBLEM REP	SUBSYS	CAUSE	IMPACT	FIX		
Development Testing								
PY 61	3.1.2	1	WO	C	M	L	3.06	18.45.07
62	3.1	1	WO	A4	M	M	3.03	18.26.67
		2	WO	A3c	M	L		
		3	WO	A3a	M	L		
63	3.6	1	WO	B1	L	L	3.03	17.01 29
		2	WO	A3e	L	L		
		3	WO	B1	L	L		
		4	WO	A3e	L	L		
		5	WO	B1	L	L		
		6	WO	A3e	L	L		
		7	WO	A3e	M	L		
		8	WO	B1	L	L		
		9	WO	A3e	L	L		
		10	WO	B1	L	L		
		11	WO	A3e	L	L		
64	3 1		WO	A4	H	H	–	–
65	3.1.4	1	IC	A2b	H	L	4.74	44.44.64
		2	IC	A2b	L	L		
		3	IC	A2b	M	L		
66			WO				3.03	13.44 32
67	4 2 2	1	IC	A2b	M	L	3.36	14.45 50
68	3 1 2		WO				3.03	23.33 89
69		1	TC	A4	M	L	5.48	1:03:39 98
70	4 2		IC				4.00	34:26 20
71		1	TC	D	M	M	3.59	29.01.03
		2	TC	D	L	L		
		3	TC	D	L	L		
		4	TC	D	L	L		
72		1	TC	A1a	M	L	3 52	30.08.42
		2	TC	A1a	M	L		
73		1	WO	A2c	M	M	3 11	23.25 29
74	4 5	1	IC	A2b	M	L	3 15	18 14 12
		2	IC	C	M	L		
		3	IC	C	M	L		

TABLE 6-3 (Continued)
FAMMES TEST RESULTS (DEVELOPMENT AND INSTALLATION CHECKOUT TESTING)

TEST RUN	UM REF	PROBLEM REP	SUBSYS	CAUSE	IMPACT	FIX	CPU TIME (SECONDS)	COMPUTER TIME
Development Testing								
PY 75		1	PM	A2c	M	M	10.19	24:24.58
		2	PM	A2c	M	M		
76		1	PM	A2c	M	M	4.06	53:12.58
		2	PM	C	L	L		
		3	PM	A2b	M	M		
		4	PM	A2b	M	M		
		5	PM	A2b	M	M		
		6	PM	A2b	M	M		
77	4.5.1	1	IC	A2b	L	L		
78			IC				13.15	1:23.18.78
79			PM				3.19	20:37.54
106		1	IC	A2b	H	M	4.03	17:20.55
		2	IC	A2b	H	L		
OnSite Testing								
TR 1		1	WO	C	H	L	3.00*	
		2	WO	A2b	M	L		
2		1	IC	A4	H	M	3.00*	
		2	IC	A4	H	M		
3		1	WO	A2e	H	L	3.00*	
4		1	WO	C	M	-	3.00*	
5		1	WO	C	L	-	2.84	13:50.00
		2	WO	A5	H	M		
6		1	WO	A2b	L	L	3.00*	
7		1	WO	C	M	-	3.22	1:58.53
8		1	WO	A2b	L	L	3.00*	
9		1	SYS	6	H	H	3.00*	
10		1	SYS	6	M	H	3.00*	
11			PM	AIF	H	L	2.68	20:40.00
12		1	PM	AID	M	L	3.00*	
13		1	PM	B1	M	L	3.00*	
14		1	PM	D	H	M	3.00*	
15		1	IC	A2b	L	L	3.00*	
		2	IC	A2b	H	M		
Test Time Preparing for AT			ALL				196.6	1:47:44.18

*ESTIMATED CPU TIME SHOWN. ACTUAL VALUE NOT RECORDED

TABLE 6-3 (CONTINUED)
FAMMES TEST RESULTS (USER OPERATIONS)

USER PR	DATED	SUBSYS	CAUSE	IMPACT	FIX	COMMENTS
1	3/02	PM	A2c	H	L	
2	4/08					NOT SW PROBLEM
3	4/19					NOT SW PROBLEM
4	5/14					NOT SE PROBLEM
5	5/29	WO	A3f	M	M	
6	6/02	PM	D	H	M	
7	6/02	WO	2a	L	M	
8	6/02	PM	2a	M	M	
9	6/30	IC	B3/D	L	L	UM CORRECTED
10	6/30	IC	B2	L	L	UM CORRECTED
11	6/30	SYS	D	L	L	
12	6/30					NOT PROBLEM
13	6/30	IC	2b	L	L	
14	6/24	WO	D	L	M	
15	6/27	WO	B1	L	L	
16	6/27	IC				NOT PROBLEM
17	6/29	IC	B1	L	L	UM CORRECTED
18	6/27	IC	B2	L	M	UM CORRECTED
19	6/27	WO	B2	L	M	UM CORRECTED
20	6/27	IC				NOT PROBLEM
21	7/02	IC	D	L	M	
22	7/10	IC	A2c	L	L	
23	7/09	IC	A2b	M	L	
24	7/09	IC				OUT OF SCOPE

TABLE 6-3 (CONTINUED)
FAMMES TEST RESULTS (USER OPERATIONS)

USER PR	DATED	SUBSYS	CAUSE	IMPACT	FIX	COMMENTS
25	7/17	WO/IC	A2b	L	M	
26	7/17	WO/IC	B2	L	L	
27	7/17	MH	A2c	L	L	
28	7/17	SYS				OUT OF SCOPE
29	7/17	SYS				OUT OF SCOPE
30	7/17	WO	B2	L	L	UM CORRECTED
31	7/16	WO	B1	L	L	UM CORRECTED
32	7/16	IC	26	L	L	
33	7/16	WO	A1a/A2b	H	H	
34	7/16	WO	B1, 2, 3	L	L	11 CHANGES TO UM
35	7/16	WO	2b	M	L	
36	7/16	TC	2b	L	L	
37	7/16	TC	B1	L	L	UM FIX
38	7/16	TC	B2	L	L	UM FIX
39	7/16	TC	B2	L	L	UM FIX
40	7/18	PM	2a	M	M	
41	7/18	MH	D	L	L	
42	7/18	TC				NOT PROBLEM
43	7/18	PM	2b	L	L	UM FIX
44	7/18	WO	D	H	M	
45	7/28	SYS				REPEAT
46	7/28	SYS				REPEAT
47	9/16	PM	B2	L	M	

TABLE 6-3 (CONTINUED) LEGEND

UM REFERENCE User Manual Reference if test run was demonstrating a capability described in Users Manual

PROBLEM REP Number of Problem Reports generated during a test run

SUBSYS Subsystem being exercised during test run

IC Inventory Control
PM Preventive Maintenance
WO Work Order Processing
MH Maintenance History
SYS System Utilities
TC Trouble Call function of WO

CAUSE Error Cause Code

A PROGRAMMER

1 Computational examples wrong variable in the equation, overflow or underflow, missing computation, unneeded computations

a Missing Equation
b Division by Zero
c Ambiguous Statement
d Erroneous Calcultion
e Unnecessary Calculation
f Mixed Mode

2 Logic Error examples missing tests, incorrect tests (wrong relational operator, sequencing of decisions)

a Missing Test
b Erroneous
c Wrong Sequence

3 Data Handling Errors examples subscript errors, variable initialization, referencing or updating the wrong variable, using the wrong arithmetic operator.

a Subscript Errors
b Argument List Inconsistencies
c Uninitialized Variables
d Data Transfer Errors
e Data Structure Errors
f Scaling/Precision Errors

4 Unknown error source

B DOCUMENTATION

1 Erroneous
2 Insufficient
3 Ambiguous
4 Violation of Coding Standard
5 Uncontrolled Change

C TESTER

D. DESIGN

IMPACT Error Impact Code

Low - The system functioned satisfactorily with minor irregularities.

Medium - The system functioned, but unsatisfactorily.

High - The system did not functioned.

Unknown - The test did not show conclusively wheter there is an error or not.

0 - No error detected.

FIX Error Repairation Code

Low - The combined analysis and correction took less than 1.5 person hours.

Medium - The combined analysis and correction ranged from 1.5 to 12 person hours.

High - The combined analysis and correction ranged from 12 to 36 person hours.

Unknown - Error repairation cost estimate not attempted.

test run. Specific CPU execution time and computer operation time was collected during development testing. Figures 6-1 and 6-2 illustrate graphically the occurrence of failures over calender time and CPU time respectively.

In summary, seventy-one (71) problem reports were reported during the testing of the system. Sixty-four (64) specific test runs/sessions were conducted to uncover these 71 problems. This data is provided in the first three pages of Table 6-3. A total of 16.34 computer operation hours were utilized during these testing sessions. Thus, since the system was 16,096 lines of executable code, the fault density at the end of the tesing was .0044. The average failure rate, using the computer operations hours expended to expose the 71 problems, was 4.34. Using the last three testing sessions, two problems were found duing 2.15 hours of testing. This calculates to a failure rate at the end of testing of .93.

After installation, during operation of the system by the users, 35 problems were reported. This number does not include additional problems reported by the user that, after analyses, were found not to be problems or were out of scope of the specification. An estimated 480 computer operation hours were utilized during the period of time these 35 problems were reported. The failure rate exhibited during user operation then was .073. Adding these additional problems to the 71 found during testing meant that a total of 106 problems had been found in the 16,096 lines of code (a fault density of .0066).

Without knowledge of this actual performance, the prediction and estimation methodology developed during this research effort was followed (see the Guidebook in Volume II). Table 6-4 summarizes the results of the application of the methodology utilizing only these prediction and estimation relationships recommended in Table 5-23.

The results were encouraging. The predicted fault density was .0063 faults per line of executable code, which was within 43% of the actual fault density using the problem reports found during testing and within 4.5% of the actual fault density using both the test problem reports and the operational problem reports. The estimated failure rate was .087 failures per operations hour, within 19% of the observed actual failure rate.

The predicted fault density was expected to be closer to the fault density calculated using only the problem reports identified during testing since the fault densities collected from the 31 data sources and used to calculate the average fault densities related to the application type, A, were primarily from formal test programs. Little data, as observed in ection 4, was available from operational systems. The results shown, however, demonstrated the predicted value to be very close to the overall fault density recorded through operation. Data collection efforts in operational environments will help correct any bias in

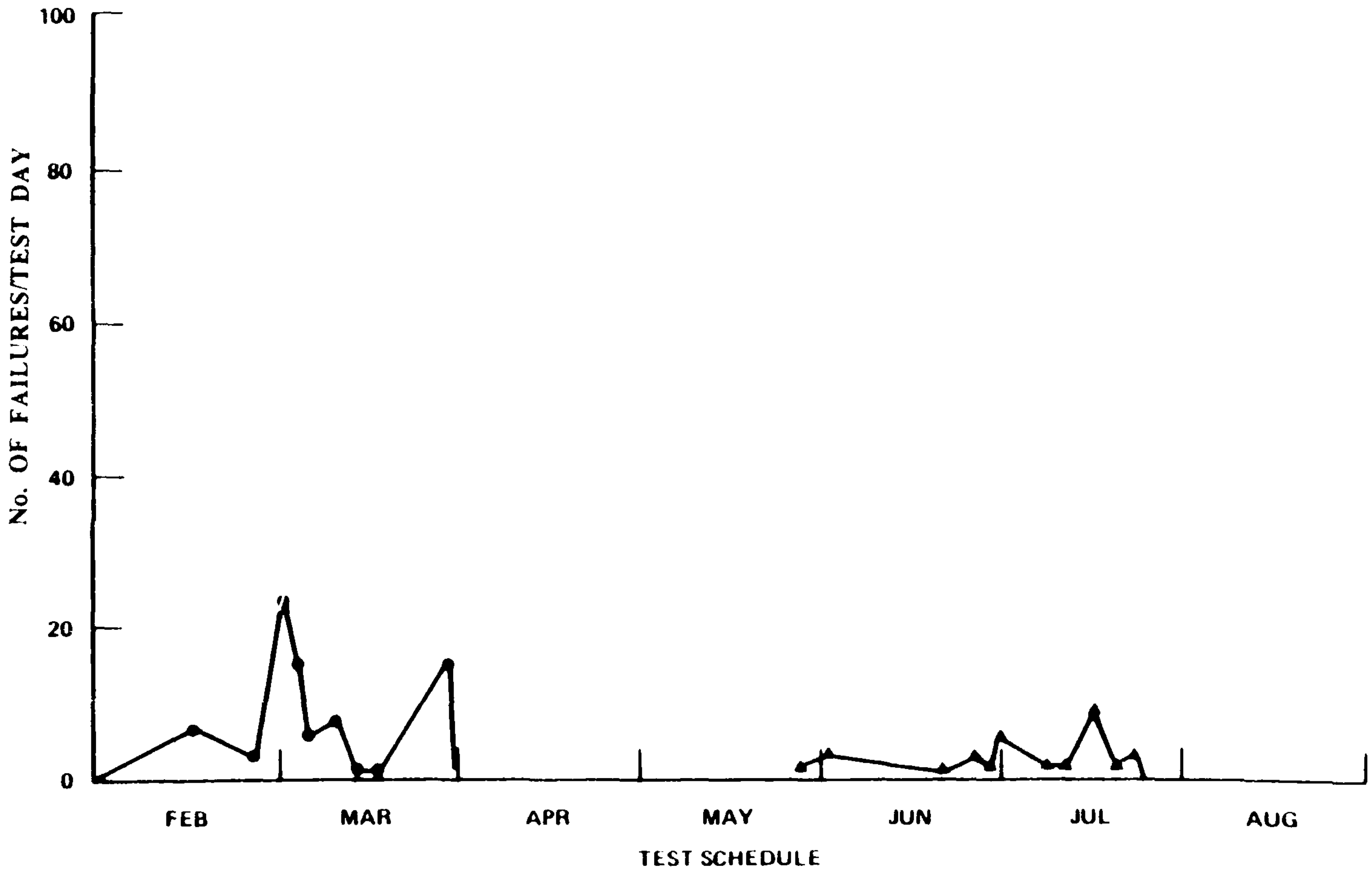


FIGURE 6-1 FAILURE DISCOVERY BY CALENDAR TIME

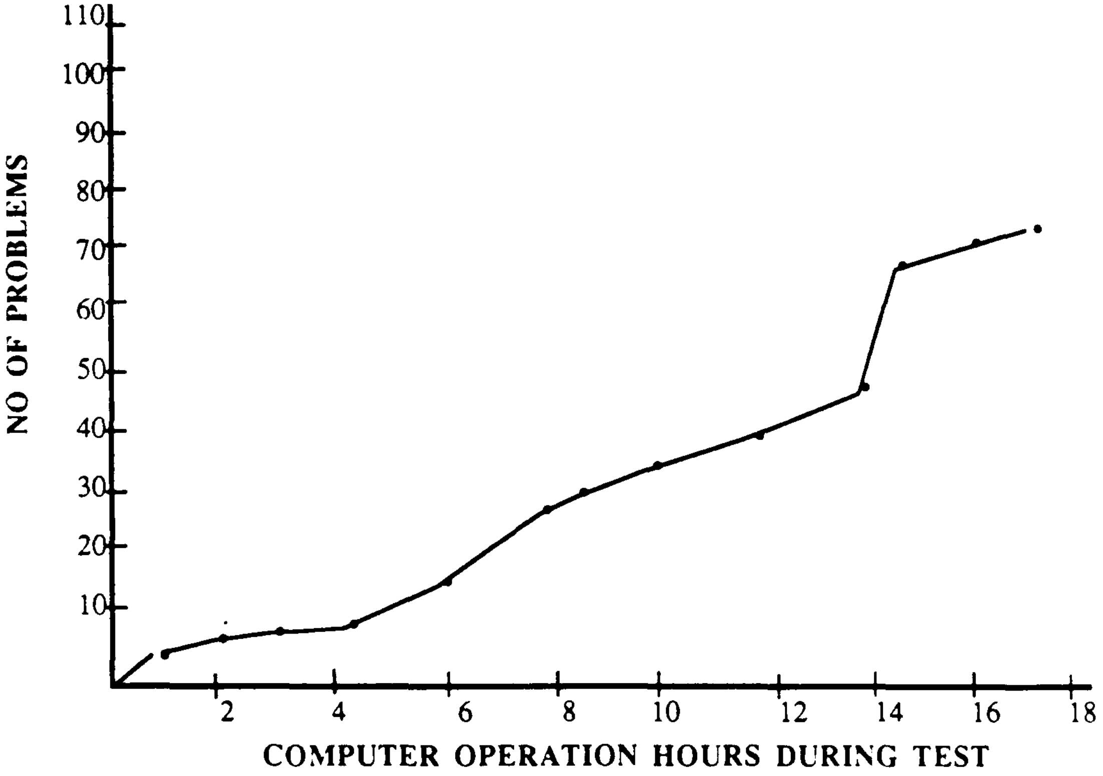


FIGURE 6-2
CUMULATIVE NUMBER OF PROBLEMS
FOUND DURING DEVELOPMENT TESTING

TABLE 6-4
METHODOLOGY APPLICATION (RECOMMENDED METRICS)

PREDICTION

RP = A * D * S

A = APPLICATION TYPE = PRODUCTION CENTER

BASE LINE FAULT DENSITY = .0085
BASE LINE FAILURE RATE = .108

A = .0085

D = DEVELOPMENT MODE = SEMI DETACHED

D = DM = (.008 DC - .04)/.013

DC = 25/39 = .64

DM = (.008 * .64 - .04)/.013

= 1.09

D = 1.09

S = SOFTWARE CHARACTERISTICS

S = SL * SX * SR

SL = FORTRAN = 1

SX = 1.5 (25) + 1(140) + .8(246)/411

= .91

SR = PR/NM

= .2 < .25 SR =.75

*S = 1 * .91 * .75 = 68*

S = .68

PR = .005 * 1.09 * .68 = .0063

Actual Fault Density at end of Test = .0044

Prediction Error = $\left|\frac{\text{RP - Actual FD}}{\text{Actual FD}}\right| \approx 43\%$

Actual Fault Density at end of 3 months operation = .0066
Prediction error = 4.5%

ESTIMATION

RE = F_{T1} * T_1

F_{T1} = Observed Average Failure Rate during Test = 4.34

T_1 = .02 * TC

TC = 1/VS = 1/1 = 1

T_1 = .02 * 1 = .087

RE = 4.34 * .02 = .087

Actual Failure rate during operations ≈ .073

Estimation Error = $\left|\frac{\text{RE - Actual FR}}{\text{Actual FR}}\right| = 19\%$

the methodology over time.

Utilizing all of the predicted and estimation relationships developed, including these not recommended because further data or analyses are required, the results are almost as good (see Table 6-5).

Taking into account the additional influences represented by these additional predictors should result in a more accurate prediction, but in this case, the prediction was less accurate (22% and 19% errors for the predicted fault density and 30% error for the estimated failure rate) in two of the three cases.

A possible rationale for the predicted fault density being high compared to the fault density at end of test is that the problems found during the design review (used as input to the Quality Review metric) are not counted as problems in the fault density calculation and these problems, identified early, were corrected then. The estimated failure rate was high probably because the metrics (in the expanded methodology) indicated that the system wasn't tested as extensively as preferred. The estimation methodology, then, modifies the estimated failure rate up because there is less confidence that the observed failure rate during test is a true representation of the system.

As stated earlier in this report, eventually we feel the prediction techniques should be predicting failure rate, like the estimation techniques, rather than fault density. The prediction techniques have been derived using fault density data from the data sources. Ignoring that fact, and simply using the prediction metrics shown in Figure 6-4 and the baseline failure rate instead of fault density, our predicted failure rate would be:

$$RP = .108 * 1.09 * .68 = .08$$

which represents only a 9.6% prediction error.

Additional data collected during this experiment are presented in Figure 6-3 and Table 6-6. In Figure 6-3, the Impact column describes the criticality of the fault to the system operation, a high impact meant the system would not function, a medium impact meant the system would operate but not satisfactorily, and a low impact meant the system would function satisfactorily with minor irregularities. Note 20% of the faults were reported during testing were judged to have a high impact on the system. The Fix column records the impact on fault repair. A high rating meant the combined analysis and correction effort took between 12 and 36 person hours to correct, a medium rating meant the repair action took between 1.5 and 12 person hours, and a low meant less than 1.5. Using average times of 24, 8, and 1, the average time to repair a fault was approximately 4 hours. Only 3 faults during testing were considered to require longer than 12 person hours. In Table 6-6, 41% of the faults found involved logic

TABLE 6-5
METHODOLOGY APPLICATION (FULL METRIC SET)

PREDICTION

RP = A • D • S

A = Application Type = Production Center

Baseline Fault Density = .085
Baseline Failure Rate = .108 A = .085

D = Development Mode = Semi-Detached

$D = D_M = (.008\ D_C - .04)/.013$

$D_M = (.008 \bullet 64 - .04)/.013$ D = 1.09
= 1.09

S = Software Characteristics

S = SA • ST • SQ • SL • SM • SR • SX

SA = Error Tolerance Checklist
Not applied SA = 1

ST = Traceability
= NR - DR/NR = .95
if $\geq$ .9 ST = 1

SQ = Quality Review
= DR/NR = 33/68
= .48 if < .5 SQ =1

SL = FORTRAN SL = 1

SM = (.9(406) + (5) = 2(0))/411
= .9 SM = .9

SR = PR/NM = .2
if < .25 SR = .75

SX = (1.5 (25) + (140) + .8(246))/411
= .91 SX = .91

S = 1 • .95 • 1 • 1 • .9 • .75 • .91 = S = .58

RP = .005 • 1.09 • .58 = .0053

Prediction Error with Actual FD after test = 22%
Prediction Error with Actual FD During Ops = 19%

ESTIMATION

$RE = F_{T1} \bullet T_1$

$F_{T1} = 4.34$

$T_1 = .02 \bullet TE \bullet TM \bullet TC$

TE = 40/AT
= 40/33 = 1.2 TE =1

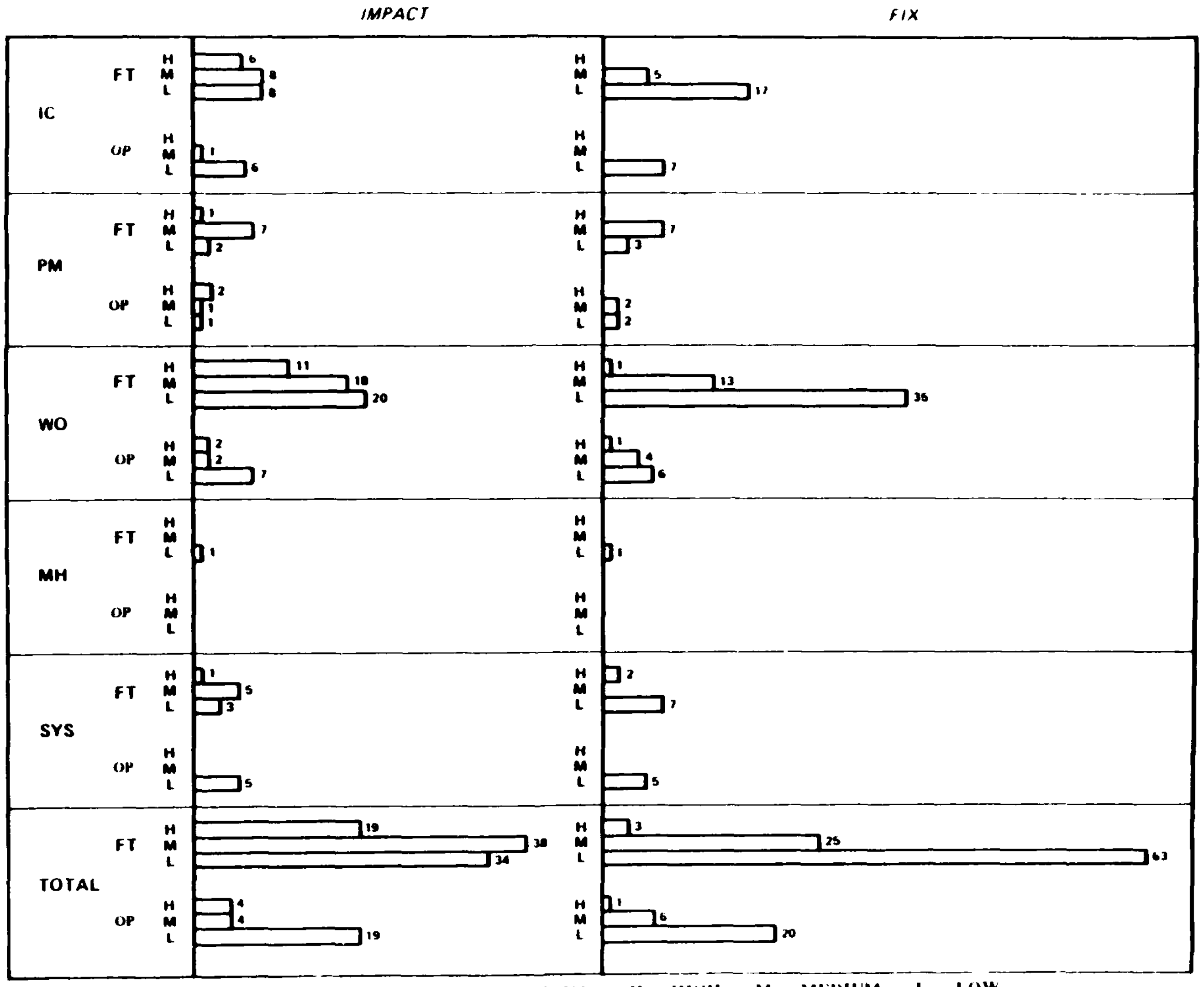
TM = TT/TU
= 3/15 = .2 TM = 1.1

TC = 1/VS
= 1/1 = 1 TC = 1

$T_1 = .02 \bullet 1 \bullet 1.1 \bullet 1 = 0.22$

RE = 4.34 • .022 = .095

ESTIMATION ERROR = 30%



LEGEND IC = Inventory Control
PM = Preventive Maintenance
WO = Work Order Processing
MH = Maintenance History
SYS = System Utilities

FT = FORMAL TEST OP = OPERATIONS H = HIGH M = MEDIUM L = LOW

*Numbers shown are number of problems recorded in each category

FIGURE 6-3 PROBLEM IMPACT AND FIX EFFORT DISTRIBUTION

TABLE 6-6 TYPES OF ERRORS

ERROR CAUSE	NUMBER OF ERRORS	
	FORMAL TEST	OPERATIONS
PROGRAMMING		
COMPUTATIONAL	5	1
LOGIC	38	9
DATA	18	1
SYSTEM	8	
INTERFACE	1	
TIMING	3	
DOCUMENTATION	5	10
TESTING	9	4
DESIGN	4	
CONFIGURATION CONTROL	1	

errors which is consistent with other data presented earlier in this report.

6.2 Assessment

The experiment confirmed two vital goals of this overall research effort:

(1) Software reliability prediction and estimation appears to be feasible. The accuracy experienced during the experiment (- 30% error) was encouraging. Further refinement of the metrics based on future data collection should improve the techniques (see Section 7 for suggested future research).

(2) The reliability prediction and estimation technology appears to have significant potential for aiding in the development of more reliable systems. Table 6-7 highlights how the predictions and estimations provide support to the development of more reliable systems.

A key idea generated or supported during the experiment was that the prediction techniques and the metrics that support them aid in identification of the parts of the system which eventually exhibit the highest fault density or failure rates. In analyzing data source 10 and 17, the metrics were generally accurate in identifying those subsystems or CSC's that contained the most faults. During the experiment, the metrics accurately predicted that Work Order Processing and the System Utilities subsystems were the most error prone (highest fault density). Further evaluation is needed to assess their prediction effectiveness at a module level. The information provided by the metrics and predictions then can be used to support software engineering decisions which typically include:

(1) Redesign of module (replacement)

(2) Decomposition of module

(3) Allocation of most experience programmer or tester

(4) Reassessment of algorithms to simplify

(5) Rework to comply with Standards

(6) Further analysis

(7) Further testing

It is in the support of these activies that the real payoff of the technology is realized, since the reliability of the software

TABLE 6-7 SOFTWARE RELIABILITY PREDICTIONS AND ESTIMATION TECHNOLOGY BENEFITS

ACTIVITY SUPPORTED	BENEFITS
REQUIREMENTS SPECIFICATION	• APPLICATION TYPE PROVIDES A BASELINE SOFTWARE RELIABILITY REQUIREMENT
PROPOSED APPROACH EVALUATION	• PREDICTION METHODOLOGY SUPPORTS AN EARLY ASSESSMENT OF IMPACT OF DEVELOPMENT APPROACH ON SOFTWARE RELIABILITY
GOVERNMENT REVIEW PROCESS SRR PDR CDR TRR	• CHECKLISTS SUPPORT EVALUATION AND PROBLEM IDENTIFICATIN AT EACH MILESTONE AND AN UPDATE OF RELIABILITY PREDICTION. ESTIMATION METHODOLOGY SUPPORTS EVALUATION OF TEST APPROACH
SOFTWARE DESIGN AND IMPLEMENTATION	• SUPPORTS GOAL SETTING BY DEVELOPERS • SUPPORTS ARCHITECTURAL DESIGN BY ASSESSING RELIABILITY IMPLICATIONS • SUPPORTS PROBLEM IDENTIFICATION AND CORRECTION STRATEGY • SUPPORTS DATA COLLECTION
TESTING	• PROVIDES DATA COLLECTION GUIDELINES • PROVIDES STANDARDS FOR PERFORMANCE AND PROGRESS ASSESSMENT • PROVIDES BASIS FOR ACCEPTANCE
POST DEPLOYMENT SUPPORT	• PROVIDES BASIS FOR ASSESSMENT OF PERFORMANCE • PROVIDES OPPORTUNITY FOR IDENTIFICATION OF RELIABILITY IMPROVEMENT WORK • SUPPORTS ASSESSMENT OF ENHANCEMENTS • PROVIDES GUIDELINES FOR DATA COLLECTION

will be improved during the development process as a result of these activities.

7.0 CONCLUSIONS/RECOMMENDED FUTURE RESEARCH

7.1 General

The primary goal of this research effort was to develop a methodology for predicting software reliability. The Guidebook in Volume II of this report provides all of the procedures for data collection, calculating the metrics, using the models and reporting to effectively apply the methodology. The methodology is based on a framework for measuring software reliability that spans the life cycle of a software system. The methodology is preliminary in nature. It provides the basis for evolution of the prediction and estimation techniques as a result of future data collection and analysis.

A key result of this effort was the data collected. A significant portion of the effort expended during the project was devoted to collecting general reliability data from a wide range of systems, detailed data from two systems, and detailed data from another system during the experimental application of the methodology.

The experiment results were promising. Accurate predictions and estimations (within 30% of actuals) were made. However, more detailed evaluations of the results are needed and more applications of the methodology are needed before practical application is recommended. This section of the report is devoted primarily to recommending what future research should be conducted.

The utility of metrics as problem indicators was further supported. Specific analyses were conducted that demonstrated the accuracy of some metrics in pinpointing problem areas in a system.

The high level reliability indicators, such as fault density and failure rate by Application Type appear to be consistent and supported intuitively. The decision to base the methodology on a baseline prediction using Application Type probably was key to results achieved. Many of the more detailed multipliers (metrics) in the methodology, however, did not perform as well as expected. The relationships derived from regression analysis were not statistically significant for many of the metrics and a more simplified table look-up approach was taken in the methodology based on the observed trends in the data. The utility of metrics to pinpoint problem modules was deomonstrated and is a promising finding. Some metrics were dropped from consideration. The theoretical foundation of the methodology, therefore, needs significant reinforcement. Many additional ideas about software reliability were generated during the project. In the following paragraphs, recommendations for future research are made. They include both efforts that will enhance and refine the methodology developed during this project and the related ideas about reliability.

7.2 Future Research Recommendations

The following research ideas are offered for consideration. They are organized as follows:

DATA COLLECTION

- Data Collection is the keystone to the evolution and refinement of the prediction and estimation methodology. Use of the data collection procedures in Appendix C of the Guidebook are recommended for use on any software developments. This is especially true for fielded systems since failure rate data is especially needed. Collection of this data by the RADC sponsored DACS and analysis of the accuracy of the methodology could follow.

- As more data is collected, the older data sources should be purged from the data base and the baseline values and metric multipliers updated.

- Additional data sources are needed in the Tactical, Process Control and Developmental application categories.

- Data from Ada projects are needed. No data was analysed from systems implemented in Ada in the current data base.

- During projects where data collection is to be performed, the data collection procedures should be contractually required and a Data Definition Document and Data Collection Guide should be required CDRL's.

PREDICTION/ESTIMATION TECHNIQUES

- As more data is collected, further analyses of the prediction and estimation techniques should be sponsored. A goal would be to have formal, statistically supported functions embedded in the methodology.

- The analyses should be done not only at a system level using the Application and Timing categorization schema but also at a function level as suggested in Section 3.

- The analyses should also be done at the unit level.

Statistical techniques valid when dealing with data where the independent variable (fault density) is often zero should be explored. Data from Data Sources 10 and 17 are available for this level of analyses.

- Other metrics should be considered. Function Points, for example, have been mentioned in the literature but were not investigated during this effort.

- Further investigation into the relationship between fault density and failure rate (called the transformation function) is recommended.

- Addition of a Section in the Guidebook that describes how to combine the Software Reliability Prediction and estimations with hardware predictions is recommended.

SOFTWARE RELIABILITY CONCEPTS

- Revisions to the Software Quality Measurement framework should be made. Those revisions should include changing the Quality factors to the following:

  Reliability
  Integrity
  Efficiency
  Usability
  Supportability
  Reusability

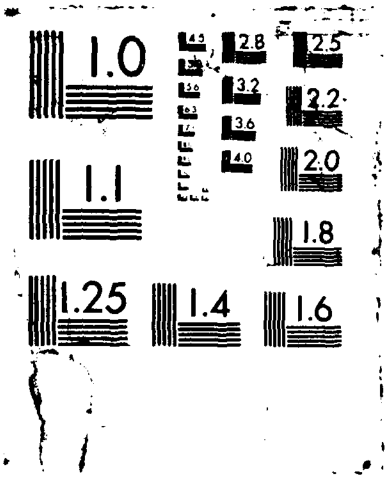
  The combination of correctness, verifiability and survivability into Reliability is recommended. Also recommended is the combination of Maintainability, Flexibility and Expandability into Supportability; and Portability [illegible] Interoperability into Reusability. This re[illegible] in factors should effect a corres[illegible] combination of criteria and metrics. The [illegible] contained in the methodology should [illegible] metrics corresponding to Reli[illegible] framework.

- A corresponding revision [illegible] Measurement System is re[illegible]

- An overall mode[illegible] role in a system [illegible] would identify [illegible] and the [illegible]

AD-A190 018 METHODOLOGY FOR SOFTWARE RELIABILITY PREDICTION VOLUME 3/3
1(U) SCIENCE APPLICATIONS INTERNATIONAL CORP SAN DIEGO
CA J MCCALL ET AL. NOV 87 RADC-TR-87-171-VOL-1

UNCLASSIFIED F30602-83-C-0118 F/G 12/5 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

schema, and environmental influences such as workload, input variability, mobility considerations, power, etc. This model would be of use to discuss the combination of software reliability concepts with these other aspects of a system so that it is taken into account in system reliability. Consideration should be given to the terms availability or dependability for software to avoid controversy with using reliability since software reliability is not a function of aging or wearout. The terms availability or dependability are more consistent with the concepts of error tolerance, robustness, recoverability, survivability and the fact that software failure is a function of latent defects and unanticipated usage. In either case, software exhibits a failure rate which must be considered in a system reliability program.

MILITARY STANDARDS

- Revisions to MIL-STD 785C are recommended to include software reliability concepts. The Guidebook in Volume II is the software equivalent to MIL-STD 756B and in part MIL-STD 785C but conceptually and practically, recognition of software in MIL-STD 756B is advised with reference to the Guidebook as a preliminary implementation guide.

OTHER

- The Guidebook should be expanded to cover software life cycle support (or Post Deployment Software Support). The equivalent hardware concepts are called logistics support. Maintainability (the time to repair) is a key issue in hardware availability concepts and should be considered in software reliability prediction and estimation as well.

- The Guidebook should be coordinated with the draft DOD Data Collection Guidebook, the DACS Software Data Collection Guidebook, and the Software Management Indicators Pamphlet and Software Quality Indicators Pamphlet being developed by AFSC.

This extensive list of recommendations is based on the promise this research provides. It acknowledges the deficiencies in the current technology but recognizes the key to improvement is through data collection and analysis.

8.0 REFERENCES

ADAM84 Edw. N. Adams, "Optimizing Preventive Service of Software Products", IBM Journal of Research & Development, pp. 2-14, Jan. 1984.

AIRT83 Air Tunnel Control System (project performed by SAI Huntsville at the Arnold Engineering Development Center) unpublished data.

AKIY71 F. Akiyama, "An Example of Software System Debugging". IFIPS Proc., 1971, p. 37.

ANGU79 J. Angus, et al, "Validation of Software Reliability Models", RADC TR-79-147, June 1979.

ANGU83 J. E. Angus, et al, "Reliability Model Demonstration Study", RADC TR-83-207, August 1983.

ANSI81 American National Standards Institute (ANSI), "American National Dictionary for Information Processing", Report X3/TR-1-81.

BAKE77 Baker, W., "Software Data Collection and Analysis: A Real-Time System Project History", RADC TR-77-192, June 1977.

BASI77 Basili, V., et al, "The Software Engineering Laboratory", University of Maryland TR-535, May 1977.

BASI81 Basili, V., et al, "A Controlled Experiment Comparing Software Development Approached", IEEE Transactions on Software Engineering, Vol. SE-7, No. 3, May 1981.

BEAU81 M. Manielle Beaudry, "A Statistical Analysis of Failures in the SLAC Computing Center", Digest. COMPCON 1979. pp. 49-52, IEEE Catalog No. 79CH1393-8C, February 1979.

BELL76 T. Bell, et al, "An Extendable Approach to Computer-Aided Software Requirements Engineering", 1976 (SEC).

BOEH81 Barry W. Boehm, Software Engineering Economics, Prentice-Hall Inc., 1981.

BOWE83 T. Bowen, "Software Quality Measurement For Distributed Systems," RADC TR-83-175.

BOWE85 T. Bowen, et al., "Specificatiion of Software Quality Attributes", RADC TR-85-37 (3 Vols), February 1985.

CARD82 Card, D. and McGarry, F., "The Software Engineering Laboratory", NASA Goddard Space Flight Center, SEL-81-104, February 1982.

CHEU81 R.C. Cheung, "A User-Oriented Software Reliability Model", IEEE Transactions on Software Engineering, Vol. SE-7, No. 1, January, 1981.

CURT79 B. Curtis, "In Search of Software Complexity", Presentation at Workshop on Quantitative Models of Software Reliability, Complexity, and Cost", IEEE, NY 1979.

DACS79 Data and Analysis Center for Software (DACS) "Quantitative Software Models", (DACS) SRR-1, March 1979.

DAVI81 E. A. Davis and P. K. Giloth, "No. 4 ESS: Performance Objectives and Service Experience". Bell Systems Technical Journal, Vol. 60 No. 6, pp. 1203-1224, August, 1981.

FAGA76 M. Fagan, "Design and Code Inspections and Process Control in the Development of Programs", IBM TR 00.2763, June 1976.

FISH79 D. Fish, M. Matsumoto, "Software Data Baseline Analysis", RADC TR-79-67, March 1979.

FRIE77 M. Fries, "Software Error Data Acquisition", RADC TR-11-130, April 1977.

GLOS84 Gloss-Soler, S., et al, "The DACS Software Engineering Data Collection Package", March 1984.

GOEL78 A.L. Goel, K. Okumoto, "Bayesian Software Prediction Models - An Imperfect Debugging Model for Reliability and Other Quantitative Measures of Software Systems", Rome Air Development Center Report, RADC-TR-78-155, vol I. August 1979.

GOEL79 A. L. Goel, "Summary of Progress on Bayesian Software Reliability Prediction Models", RADC TR-78-155, 1978.

GOEL83 A. Goel, "A Guidebook for Software Reliability Assessment", Syracuse TR 83-11, April 1983.

GRAS82 J. Gras and I. Hamburg, "Collection and Analysis of Data From Various Software Metrics and Reliability Estimation", Dynaflow Software Systems, Amsterdam, Netherlands, 1982.

HALS77 M. Halstead, Elements of Software Science, Elsevier Computer Science Library, NY 1977.

HECH77 H. Hecht, W.A. Sturm and S. Trattner, "Reliability Measurement During Software Development", NASA Langley CR-145205, Sep. 1977

HECH79 H. Hecht, "Fault Tolerant Software", IEEE Transactions on Reliability, Vol. R-28, No. 3, August 1979.

HECH83 H. Hecht and M. Hecht, "Trends in Software Reliability for Digital Flight Control", NASA Ames Research Center, April 1983.

HERN83 M. Herndon, et al, "The Requirements Management Methodology: A Measurement Framework for Total System Reliability Conference, December 1983.

HIER86 V. Hiering and D. Bennett, "A Developer's Perspective on Software Quality Metrics", IEEE Communications Magazine, Vol. 24, No. 9, pp 66-11, September 1986.

IEEE82 IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1982, The Institute of Electrical and Electronics Engineers, New York, NY.

IEEE83 IEEE Computer Society, "Digest of Papers, Spring COMPCON' 83", IEEE Cat No. 83CH1856-4, particularly Session 1, "Practical Approaches to Highly Available Systems", pp. 1-18, March 1983.

IYER81 R.K. Iyer, Steven E. Butner, Edw. J. McCluskey, "An Exponential Failure/Load Relationship: Results of a Multi-Computer Statistical Study" Computer Systems Laboratory, Stanford University Center for Reliable Computing, July 1981.

IYER83 R.K. Iyer and Paola Velardi, "A Statistical Study of Hardware Related Software Errors in MVS", Stanford University Center for Reliable Computing, October 1983.

JCL81 Joint Logistics Commanders' Orlando I Conference; 1981.

LEHM82 M. Lehman, "Report on Professor Lehman's Visit to St. Louis", documents by Col. Hogan, USA Material Development and Readiness Command, ALMSA, April 1982.

LIPO79 M. Lipow, Ed, IEEE Transactions on Reliability, Volume R-28, No. 3, August 1979.

LIPO83 Failure data taken from a classified/proprietary computer supplied by Myron Lipow.

LITT74 B. Littlewood & J.L. Verrall, "A Bayesian Reliability Model with a Stochastically Monotone Failure Rate", IEEE Transactions on Reliability, Vol. R-23, pp. 108-114, June 1974.

LITT80 B. Littlewood, "What Makes a Reliable Program - Few Bugs or a Small Failure Rate?", Proc 1980 NCC, p. 707-715, May 1980.

MACK83a D. A. MacKall, V. A. Regenie, and M. Gordoa, "Qualification of AFTI/F-16 Digital Flight Control System", NAECON Paper 324, May 1983.

MACK83b D.A. Mackall, "AFTI/F-16 Digital Flight Control System Experience", First Annual NASA Aircraft Controls Workshop, Langley Research Center, October 1983.

MAXW78 F.D. Maxwell "The Determination of Measures of Software Reliability" NASA-CR-158960 Final Report, NASA Langley Research Center, Hampton, Virginia, 1978.

MCCA76 T. J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, 1976.

MCCA77 J. McCall, et al, "Factors in Software Quality", RADC TR-77-369, November 1977.

MCCA80 J. McCall, et al, "Software Quality Measurement Manual". RADC TR-80-109, April 1980.

MCCA84 J.A. McCall et al, "Methodology for Software and System Reliability Prediction" Phase I Final Report. Prepared for RADC, Science Applications Inc., July 1984.

MEND79 K. Mendis and M. Gollis, "Categorizing and Predicting Errors in Software Programs", AIAA Conference Proceedings, 1979.

MIYA- I. Miyamoto, "Software Reliability in On-line Real Time Environment", Nippon Electric Co., Tokyo, Japan, (date unknown).

MOTL76 R. Motley, "Statistical Prediction of Programming Errors", RADC TR-77-175, May 1977.

MORA76 P. Morands, "Quantitative Methods for Software Reliability Measurements", Defense Communications Center, December 1976.

MUSA75 J. Musa, "A Theory of Software Reliability and its Applications", IEEE Transactions on Software Engineering. Vol. DE-1, No. 3, pp. 212-327, September 1975.

MUSA79 John Musa, "Validity Of The Execution Time Theory of Software Reliability", IEEE Transactions on Reliability, Vol. R-28, No. 3, pp. 181-191, August 1979.

MUSA80 John Musa, "The Measurement and Management of Software Reliability", Proc. IEEE, Vol. 68, September 1980, pp. 1031-1043.

MYER76 G. J. Myers, Software Reliability: Principles and Practices, John Wiley & Sons, NY 1976.

NAGE82 Phyllis M. Nagel and James A. Skrivan, "Software Reliability: Repetitive Run Experimentation and Modeling", NASA CR-165836, February 1982.

NELS78 Richard Nelson, "Software Data Collection and Analysis", Draft, RADC, September 1978.

PRES80 Jacques Press, "Computer Utilization at Several Enroute Air Traffic Control Centers", Report ARD-140-1-81. Federal Aviation Administration, December 1980.

PRES84 E. Presson, "Software Test Handbook", RADC TR-84-53, March 1984.

PRIC77 Reference Manual - PRICE Software Model, RCA PRICE Systems, Cherry Hill, NJ, December 1977.

RICH83 G. Richeson, "Reliability of Shuttle Mission Control Center Software", Johnson Space Center TR, 1983.

ROCK81 Rockwell-Collins, "Software Error Study", Contract NASZ-27495, Collins Avionics Division, 1981.

ROSS82 D.J. Rossetti and R.K. Iyer, "Software Related Failures in the IBM 3081: A Relationship with System Utilization", Stanford University, Center for Reliable Computing, CRC Technical Report 82-8, July 1982

SANA82 San Antonio, R., et al, "Application of Software Metrics During Early Program Phases", COMPSAC 82.

SDS83 DoD-STD-SDS, Defense System Software Development, Proposed MIL-STD, 12/5/83.

SEL83 Software Engineering Laboratory, Software Engineering Laboratory (SEL) Data Base Organization and User's Guide, Revision 1, NASA Goddard Space Flight Center, Greenbelt MD, SEL 81-102, March 1983 (describes data base containing data on many NASA support systems).

SHOO77 M. Shooman, S. Natarajan, "Effect of Manpower Deployment and Bug Generation on Software Error Models", RADC TR-76-400, January 1977.

SHOO83 Martin L. Shooman, George Richeson, "Reliability of Shuttle Mission Control Center Software", NASA paper, 1983.

SOIS85 E. Soistman, et al, "Combined Hardware/Software Reliability Prediction Methodology, RADC TR-85-228, 1985.

SQAM83 MIL-STD-SQAM, Software Quality Assessment and Measurement, Proposed MIL-STD, 101/82.

SSCS83 Social Security Computing System, (TBD).

STAR83 Software Technology for Adaptable, Reliable Systems (STARS) Program Strategy, DoD, 4/1/83.

SUKE76 A. Sukert, "A Software Reliability Modeling Study", RADC TR-76247, August 1976.

SUKE77 A. Sukert, "An Investigation of Software Reliability Models," 1977 RAMS Proc, January 1977.

TEIC76 D. Teichroew, "PSL/PSA A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," 1976 (SEC).

THAY76 T. Thayer, et al, "Software Reliability Study", RADC TR-76-238, August 1976.

THIB84 R. Thibodean, "Software Reliability Benchwork, RADC TR, 1984.

TROY86 R. Troy and Y. Romain, "A Statistoca; Methodology for the Study of the Software Failure Process and its Application to the ARGOS Center", IEEE Transcript on Software Engineering, Vol. SE-12, No. 9, pp 968-978, September 1986.

TURN81 C. Turner, et al, "The NASA/SEL Data Compendium", DACS, April 1981.

WAGO73 W.L. Wagoner "Final Report on a Software Reliability Measurement Study" Distribution Statement B, Technology Division, The Aerospace Corporation, August 15, 1973.

WEIS78 D. Weiss, "Evaluating Software Development by Error Analysis: The Data from the Architecture Research Laboratory, NTIS AD/A-062 922, December 1978.

WILL77 H.E. Willman, et al, "Software Systems Reliability: A Raytheon Project History", RADC-TR-77-188, June 1977.

APPENDIX A

DEFINITIONS AND TERMINOLOGY

This appendix presents definitions of the principal terms and concepts used in this report. Where possible, the definitions are taken from established dictionaries or from the technical literature. Where a rationale for the selection or formulation of a definition seems desirable, it is provided in an indented paragraph following the definition. The sources for the definitions will be found in the list of references at the end of this Guidebook.

ERROR - A discrepancy between a computed observed, or measured value or condition and the true, specified, or theoretically correct value or condition. [ANSI81]

> This definition is listed as (1) in the American National Dictionary for Information Systems. Entry (2) in the same reference states that error is a "Deprecated term for mistake". This is in consonance with [IEEE83] which lists the adopted definition as (1) and lists as (2) "Human action that results in software containing a fault. Examples include omission or misinterpretation of user requirements in a software specification, incorrect translation or omission of a requirement in a design specification. This is not a preferred usage."

FAILURE - The inability of a system or system component to perform a required function with specified limits. A failure may be produced when a fault is encountered. [IEEE83]

> This definition is listed as (2) in the cited reference which lists as (1) "The termination of the ability of a functional unit to perform its required function" and as (3) "A departure of program operation from program requirements". Definition (1) is not really applicable to software failures because these may render an incorrect value on one iteration but correct values on subsequent ones. Thus, there is no termination of the function in case of a failure. Definition (3) was considered undesirable because it is specific to the operation of a computer program and a more system-oriented terminology is desired for the purposes of this study.

FAULT - An accidental condition that causes a functional unit to fail to perform its required function. [IEEE83]

> This definition is listed as (1) in the cited reference which lists as (2) "The manifestation of an error (2) in software. A fault, if encountered, may cause a failure". Error (2) is

identified as synonymous with "mistake". Thus this definition states that a fault is the manifestation in software of a (human) mistake. This seems less relevant than the identification of a fault as the cause of a failure in the primary definition. It is recognized that the presence of a fault will not always or consistently cause a unit to fail since the presence of a specific environment and data set may also be required (see definition of software reliability).

MISTAKE - A human action that produces an unintended result. [ANSI81]

SOFTWARE QUALITY FACTOR - A broad attribute of software that indicates its value to the user, in the present context equated to reliability. Examples of software quality factors are maintainability, portability, as well as reliability. May also be referred to simply as factor or quality factor. [Based on MCCA80]

SOFTWARE QUALITY METRIC - A numerical or logical quantity that measures the presence of a given quality factor in a design or code. An example is the measurement of size in terms of lines of executable code (a quality metric). May also be referred to simply as metric or quality metric. A single quality factor may have more than one metric associated with it. A metric typically is associated with only a single factor. [Based on MCCA80]

SOFTWARE RELIABILITY - The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered. [IEEE83]

This definition is listed as (1) in the IEEE Standard Glossary. An alternate definition, listed as (2), is "The ability of a program to perform a required function under stated conditions for a specified period of time." This definition is not believed to be useful for the current investigation because (a) it is not expressed as a probability and therefore cannot be combined with hardware reliability measures to form a system reliability measure, and (b) it is difficult to evaluate in an objective manner. The selected definition fits well with the methodology for software reliability studies which will be followed in this study, particularly in that it emphasizes that the presence of faults in the software as well as the inputs and conditions of use will affect reliability.

SOFTWARE RELIABILITY MEASUREMENT - The life-cycle process of establishing quantitative reliability goals, predicting, measuring, and assessing the progress and achievement of those goals during the development, testing, and O&M phases of a software system.

SOFTWARE RELIABILITY PREDICTION - A numerical statement about the reliability of a computer program based on characteristics of the design or code, such as number of statements, source language or complexity. [HECH77]

Software reliability prediction is possible very early in the development cycle before executable code exists. The numeric chosen for software reliability prediction should be compatible with that intended to be used in estimation and measurement.

SOFTWARE RELIABILITY ESTIMATION - The interpretation of the reliability measurement on an existing program (in its present environment, e.g., test) to represent its reliability in a different environment (e.g., a later test phase or the operations phase). Estimation requires a quantifiable relationship between the measurement environment and the target environment. [HECH77]

The numeric chosen for estimation must be consistent with that used in measurement.

SOFTWARE RELIABILITY ASSESSMENT - Generation of a single numeric for software reliability derived from observations on program execution over a specified period of time. Defined sections of the execution will be scored as success or failure. Typically, the software will not be modified during the period of measurement, and the reliability numeric is applicable to the measurement period and the existing software configuration only. [HECH77]

The statement about not modifying the software during the period of measurement is necessary in order to avoid committing to a specific model of the debugging/reliability relation. In practice, if the measurement interval is chosen so that in each interval only a small fraction of the existing faults are removed, then the occurrence of modifications will not materially affect the measurement.

PREDICTIVE SOFTWARE RELIABILITY FIGURE-OF-MERIT (RP) - A reliability number (fault density) based on characteristics of the application, development environment, and software implementation. The RFOM is established as a baseline as early as the concept of the system is determined. It is then refined based on how the design and implementation of the system evolves.

RELIABILITY ESTIMATION NUMBER (RE) - A reliability number (failure rate) based on observed performance during test conditions.

FUNCTION - A specific purpose of an entity or its characteristic action. [ANSI81] A subprogram that is invoked during the evaluation of an expression in which its name appears and that returns a value to the point of invocation. Contrast with subroutine. [IEEE83]

MODULE - A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine. [ANSI81] A logically separable part of a program. [IEEE83]

SUBSYSTEM - A group of assemblies or components or both combined to perform a single function. [ANSI73] In our context, a subsystem is a group of modules interrelated by a common function or set of functions. Typically identified as a Computer Program Configuration Item (CPCI) or Computer Software Configuration Item (CSCI). A collection of people, machines, and methods organized to accomplish a set of specific functions. [IEEE83] An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions. [IEEE83] A group or subsystem united by some interaction and interdependence, performing many duties but functioning as a single unit. [ANSI73]

SYSTEM - In our context, a software system is the entire collection of software modules which make up an application or distinct capability. Along with the computer hardware, other equipment (such as weapon or radar components), people and methods the software system comprises an overall system.

☆U.S. GOVERNMENT PRINTING OFFICE: 1988-511-117/64005